Projecte Fi de Carrera

Construcció d'una API de descodificació de vídeo per a dispositius Android

Enrique Casas Gallegos

Director: Jordi García Almiñana

Departament: Arquitectura de computadors

Data: Barcelona, juny de 2013

DADES DEL PROJECTE

Títol del Projecte: Construcció d'una API de vídeo per a dispositius Android

Nom de l'estudiant: Enrique Casas Gallegos

Titulació: Enginyeria en Informàtica

Crèdits: 37,5

Director/Ponent: Jordi García Almiñana

Departament: Arquitectura de computadors

MEMBRES DEL TRIBUNAL *(nom i signatura)*

Director: Jordi García Almiñana

Vocal: Juan Luis Esteban Ángeles

President: Juan José Costa Prats

QUALIFICACIÓ

Qualificació numèrica:

Qualificació descriptiva:

Data:

Índex de continguts

Llista de taules

Llista de figures

Llista de codi

1. Introducció

Actualment s'ha incrementat l'ús de dispositius mòbils (tabletes, telèfons, etc.) per part de tot tipus d'usuaris, quasi cada dia surt un nou model de dispositiu mòbil i tots ells tenen una cosa en comú: la bateria del dispositiu dura només un dia i, fins i tot, depenent de l'ús, dura menys. Aquests dispositius mòbils tenen uns recursos molt limitats i totes les possibilitats tecnològiques que ofereixen consumeixen aquests recursos, a més possibilitats, més consum de recursos. Un cas especial és la descodificació de vídeo, ja que els nous dispositius ofereixen una qualitat de vídeo cada vegada més gran, i això requereix un consum de recursos per tal de descodificar aquest vídeo cada vegada més gran i per tant una duració de la bateria molt més curta. A part de no poder utilitzar aquests recursos per altres tasques. D'aquesta problemàtica ve la motivació per la realització d'aquest projecte.

Un altre problema que s'intentarà solucionar en aquest projecte és el fet de que hi ha desenes de fabricants i cadascun d'aquests fabricants té desenes de models de dispositius mòbils i cada model té moltes versions diferents. Això implica que costa mantenir un estàndard apropiat per la descodificació de vídeo. Aquest inconvenient se li afegeix la diversitat de formats de compressió de vídeo existent al mercat actual.

La finalitat d'aquest projecte és entendre com funciona el sistema Android i com funciona un conjunt de crides anomenades OpenMAX que ens permetran minimitzar els recursos utilitzats sobretot l'ús del processador general per descodificar vídeo, ja que podrem utilitzar un xip específic del dispositiu mòbil anomenat descodificador, per tant el processador d'àmbit general quedarà lliure per fer altres tasques del dispositiu i de retruc disminuirem considerablement el consum de bateria. La problemàtica principal d'aquestes crides és la seva complexitat per utilitzar-les ja que requereixen un alt nivell de coneixement sobre programació. Una vegada que entenem el funcionament d'aquestes crides construirem un interfície per tal de que els programadors puguin utilitzar OpenMAX de manera senzilla i transparent.

De la quantitat de sistemes operatius que hi ha al mercat s'ha escollit el sistema operatiu Android. Algunes de les raons per haver escollit el sistema operatiu Android són: per la millor accessibilitat al codi font comparat amb altres sistemes operatius, per ser gratuït, per la seva gran quota de mercat actual i per la gran comunitat de desenvolupadors que hi ha al darrera.

A continuació es mostraran amb més detall els objectius marcats per a la realització d'aquest projecte.

1.1 Objectius

Aquest projecte té diversos objectius de diferent índole, per tant els dividirem en tres tipus d'objectius: objectius generals, objectiu principal i objectius secundaris.

Els objectius generals o personals són aquells objectius pels quals vaig triar aquesta temàtica concreta per realitzar aquest projecte, aquest objectius tenen un caràcter molt general i són els següents:

- Entendre el sistema d'Android a fons, com funciona tots els seus components i com accedir a elements interns del dispositiu.
- Entendre com funciona la descodificació de vídeo i perquè és una de les tasques més costoses en quant a consum de recursos del sistema.
- Contribuir a la reducció del consum dels recursos dels dispositius mòbils.
- Contribuir a la comunitat de desenvolupadors de codi Android.

L'objectiu principal del projecte es va cercar a la primera fase del projecte, quan només era una idea i es va seleccionar degut als meus objectius generals i personals.

- Crear una interfície de programació (API) que permet als desenvolupadors d'aplicacions multimèdia d'Android utilitzar el xip específic dedicat a la descodificació de vídeo per tal d'alliberar recursos generals del sistema, millorant el rendiment de l'aplicació i del dispositiu en general i disminuint el consum de la bateria. Aquesta API ha de ser estàndard per tots els dispositius existents al mercat i l'ús d'OpenMAX ha de ser transparent al desenvolupador. A més l'API ha d'estar dissenyada per adaptar-se a nous de dispositius.

Els objectius secundaris han sorgit durant el transcurs del projecte o es van planificar a l'inici del projecte com objectius necessaris per tal de complir l'objectiu principal del nostre projecte. Els objectius secundaris són els següents:

- Estudiar i entendre el funcionament del sistema operatiu Android.
- Conèixer quin hardware hi ha darrera d'un dispositiu mòbil.
- Preparar l'entorn de desenvolupament necessari, tant per a la creació d'aplicacions

d'usuari, com per a treballar a nivell de codi nadiu.

- Estudiar, comprendre i provar com funciona OpenMAX, les diferents versions que tenen i les diferents capes existents a la seva arquitectura. Associat aquest objectiu és important entendre quina problemàtica sorgeix d'aquestes crides.
- Estudiar com OpenMAX es pot portar a Android i com aquestes crides s'adapten al entorn del sistema operatiu Android.
- Estudiar i entendre com les aplicacions actuals utilitzen el processador d'ús general. Per aquesta tasca entrarem en detall en els possibles còdecs que només utilitzin la unitat de processament general.
- Cercar la manera de crear un reproductor de vídeo lleuger que es pugui utilitzar per fer les nostres proves, en aquest objectiu es veurà una mica d'OpenGL.
- Estudiar i entendre el funcionament d'un protocol de compressió de vídeo. En aquest projecte ens centrarem en l'estàndard H264.
- Demostrar amb números reals i fets que s'aconsegueix una millora en el rendiment del nostre dispositiu si s'utilitza la unitat de descodificació especifica (GPU).

1.2 Motivació

La motivació per escollir aquest projecte ha sigut deguda al continu creixement exponencial de l'ús dels dispositius mòbils i del seus usos multimèdia. Com a futur enginyer informàtic estic a favor del software lliure i per això volia contribuir a la comunitat de software lliure d'alguna manera. Cal dir que aquesta comunitat és la que m'ha ajudat en el transcurs dels meus estudis i sobretot al desenvolupament d'aquest projecte i espero i desitjo poder ajudar en un futur pel que fa a desenvolupament d'aplicacions multimèdia.

La meva elecció a l'hora de triar un projecte que tractés sobre els dispositius Android, va ser al cursar la assignatura "Projecte de xarxes de comunicació", on vaig crear una aplicació per sistemes Android que permetia reproduir vídeos sense compressió, ja que no entrava a l'àmbit de la assignatura. Aquesta assignatura i els resultats obtinguts van aconseguir picar-me la curiositat per un futur desenvolupar d'algun tipus de biblioteca que facilités aquesta tasca.

Una altra raó és que quan em vaig plantejar el projecte vaig veure una quantitat important d'ofertes de treball buscant especialistes en aquest camp. I això hem va encoratjar encara més a començar aquest projecte.

1.3 Planificació

La planificació del projecte és una de les fases més importants d'un projecte per tal de poder establir els objectius a complir en un temps específic. Per fer la planificació d'aquest projecte es va aproximar la feina segons el nombre de crèdits a realitzar en relació a la normativa vigent de la Facultat d'Informàtica de Barcelona i es va fer una estimació de quantes hores cal per desenvolupar cada tasca concreta. En la següent taula es mostra les hores estimades que cal dedicar per finalitzar cada tasca per finalitzar el projecte:

Tasca a realitzar	**Hores**
Estudi estructura d'una aplicació Android	10
Estudi programació codi nadiu d'Android	20
Estudi elements principals d'Android	20
Estudi elements principals del dispositiu	20
Instal·lació i configuració entorn de desenvolupament	20
Estudi OpenMAX	40
Estudi OpenMAX a Android	30
Estudi problemàtica d'OpenMAX	20
Estudi de solucions	20
Creació aplicació d'ús GPU	60
Creació aplicació d'ús CPU	60
Comparació i conclusions	30
Especificació de la nova API per descodificar	20
Disseny de la nova API per descodificar	40
Implementació de la nova API per descodificar	180
Proves de la nova API per descodificar	50
Preparació de la documentació	50
Revisió de la documentació	10
Total	**700**

Taula 1: Aproximació d'hores per finalitzar el projecte

Com es pot observar al següent gràfic, tot i que l'estudi previ dels components del projecte i dels components del propi sistema d'Android tenen associades moltes tasques a complir, aquestes tasques es compleixen relativament en poc temps i el gruix d'hores dedicades es reparteixen a la fase de desenvolupament de les aplicacions i biblioteques associades.

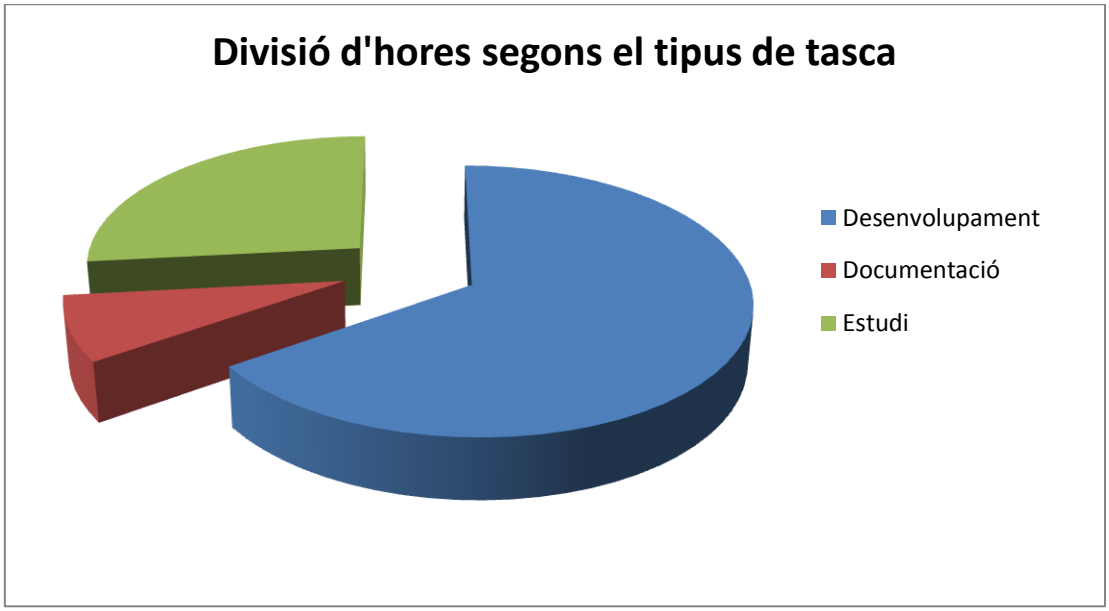

Figura 1: Divisió d'hores segons el tipus de tasca

Cal dir que aquesta planificació és una estimació sense inconvenients temporals i per tant un esquema totalment idíl·lic, donat que aquest projecte ha de complir un nombre determinat de crèdits i que aquests crèdits equivalen a 750 hores de projecte. Com podem veure la suma total d'hores previstes és de 700. Això es degut a les desviacions originades per possibles imprevistos durant el transcurs del projecte. Tots els projectes acaben tenint desviacions pel que fa al temps invertit i aquest projecte no ha sigut una excepció.

S'ha de dir que al principi d'aquest projecte es va fer una estimació dels possibles riscos que podien sorgir durant el seu desenvolupament, riscos que podien portar a desviacions d'hores planificades i invertides.

El risc més evident va ser la possibilitat d'augmentar les hores de treball a la part d'implementació degut a que es tracta d' una tecnologia desconeguda per la meva part i difícil d'utilitzar.

Un altre risc que es va tenir en compte, tot i que al final no ha afectat, va ser la no compatibilitat del dispositiu de proves amb OpenMAX. Això hagués implicat un canvi de dispositiu i la repetició de les tasques d'estudi prèvies d'aquest dispositiu.

La següent taula mostra les desviacions totals, detallades per cada tasca.

Tasca a realitzar	Inicial	Desviacions	Final
Estudi estructura d'una aplicació Android	10	0	10
Estudi programació codi nadiu d'Android	20	0	20
Estudi elements principals d'Android	20	0	20
Estudi elements principals del dispositiu	20	0	20
Instal·lació i configuració entorn de desenvolupament	20	0	20
Estudi OpenMAX	40	0	40
Estudi OpenMAX a Android	30	0	30
Estudi problemàtica d'OpenMAX	20	20	40
Estudi de solucions	20	0	20
Creació aplicació d'ús GPU	60	-20	40
Creació aplicació d'ús CPU	60	0	60
Comparació i conclusions	30	10	40
Especificació de la nova API per descodificar	20	10	30
Disseny de la nova API per descodificar	40	0	40
Implementació de la nova API per descodificar	180	10	190
Proves de la nova API per descodificar	50	10	60
Preparació de la documentació	50	0	50
Revisió de la documentació	10	0	10
Total:	**700**	**40**	**740**

Taula 2: Hores invertides per finalitzar el projecte

La disminució de 20 hores a la creació de l'aplicació d'ús de la GPU ha sigut deguda a la facilitat d'ús de la interfície d'Android a la versió 4.0. Que com veurem més endavant no és completa perquè no estan implementades moltes funcionalitats, però és suficient per complir el nostre objectiu.

L'augment d' hores de les altres tasques es degut a la complicació de l' ús d'OpenMAX que, tot i que s'havia tingut en compte com a possible risc per a la desviació d'hores, ha resultat que la previsió no va ser suficientment acurada. Això ha implicat invertir més temps de desenvolupament a les aplicacions i biblioteques que afecten a OpenMAX.

Una vegada calculades les hores que ens fan falta per completar els nostres objectius, poden fer el càlcul en mesos de la duració d'aquest projecte. Cal dir que per raons personals no he pogut dedicar a temps complet el desenvolupament d' aquest projecte. Per tant a la taula següent es mostra una planificació en dies, tenint en compte aquest factor.

Crèdits	Crèdit/hora	Hores	Hores/dia	Dies	Mesos
37,5	20	750	2	375	12,5

Taula 3: Estimació de dies per acabar el projecte

El projecte es va començar al Maig del 2012 i havent realitzat els càlculs per acabar-lo a finals de Juny del 2013 amb 1 mes de marge per les vacances d'estiu.

Una vegada s'ha tingut en compte la duració en hores de les tasques i quants dies hem d'invertir en total, s'ha pogut crear una diagrama de Gantt per tal de poder repartir aquestes tasques al transcurs de l'any.

A la següent imatge es mostra el diagrama de Gantt complet i després es mostren 3 imatges amb el diagrama de Gantt parcials per tal de veure'l més en detall.

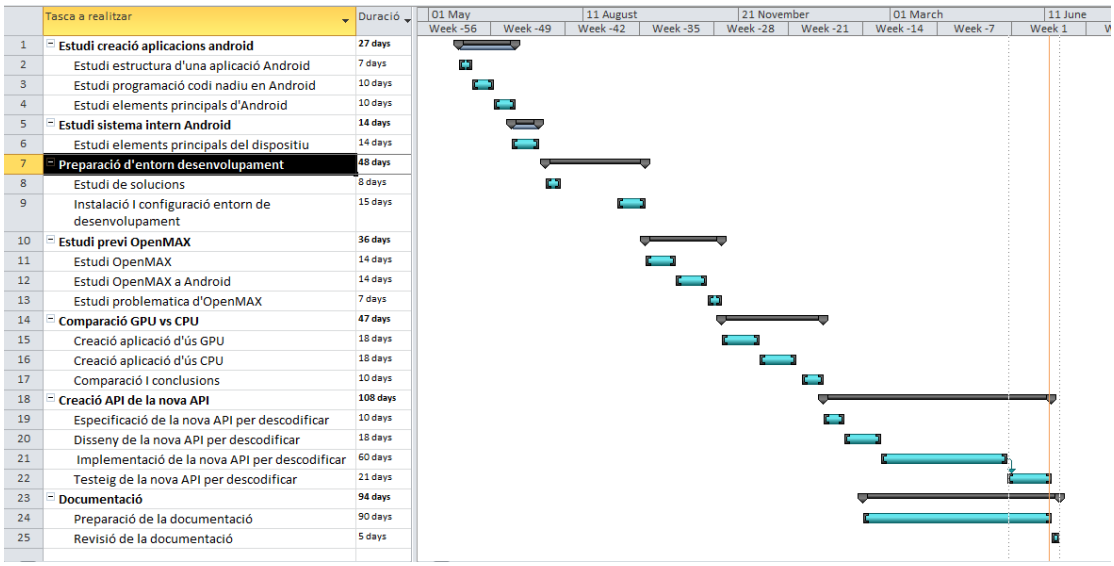

Figura2: Diagrama de Gantt complet

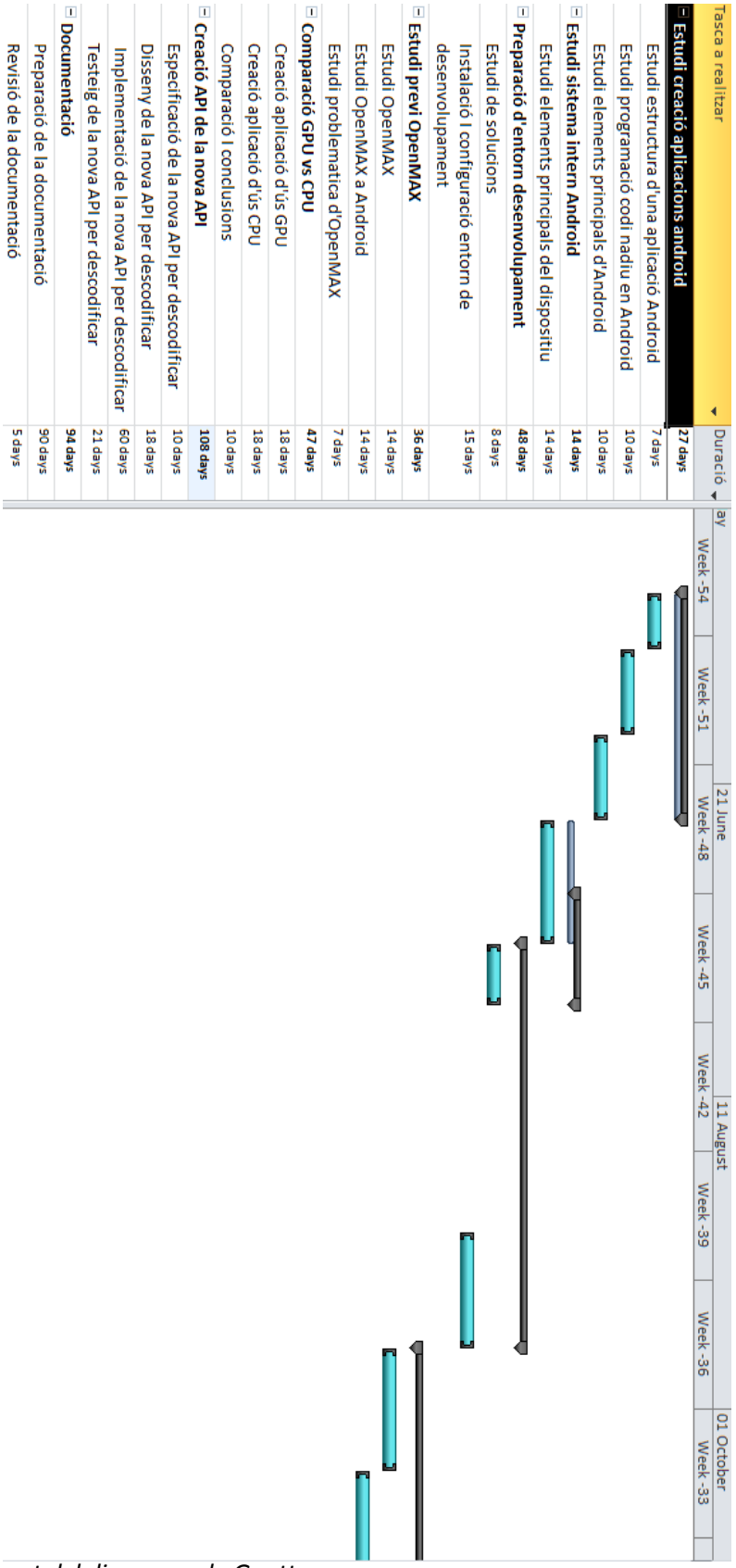

Figura 3: Primera part del diagrama de Gantt

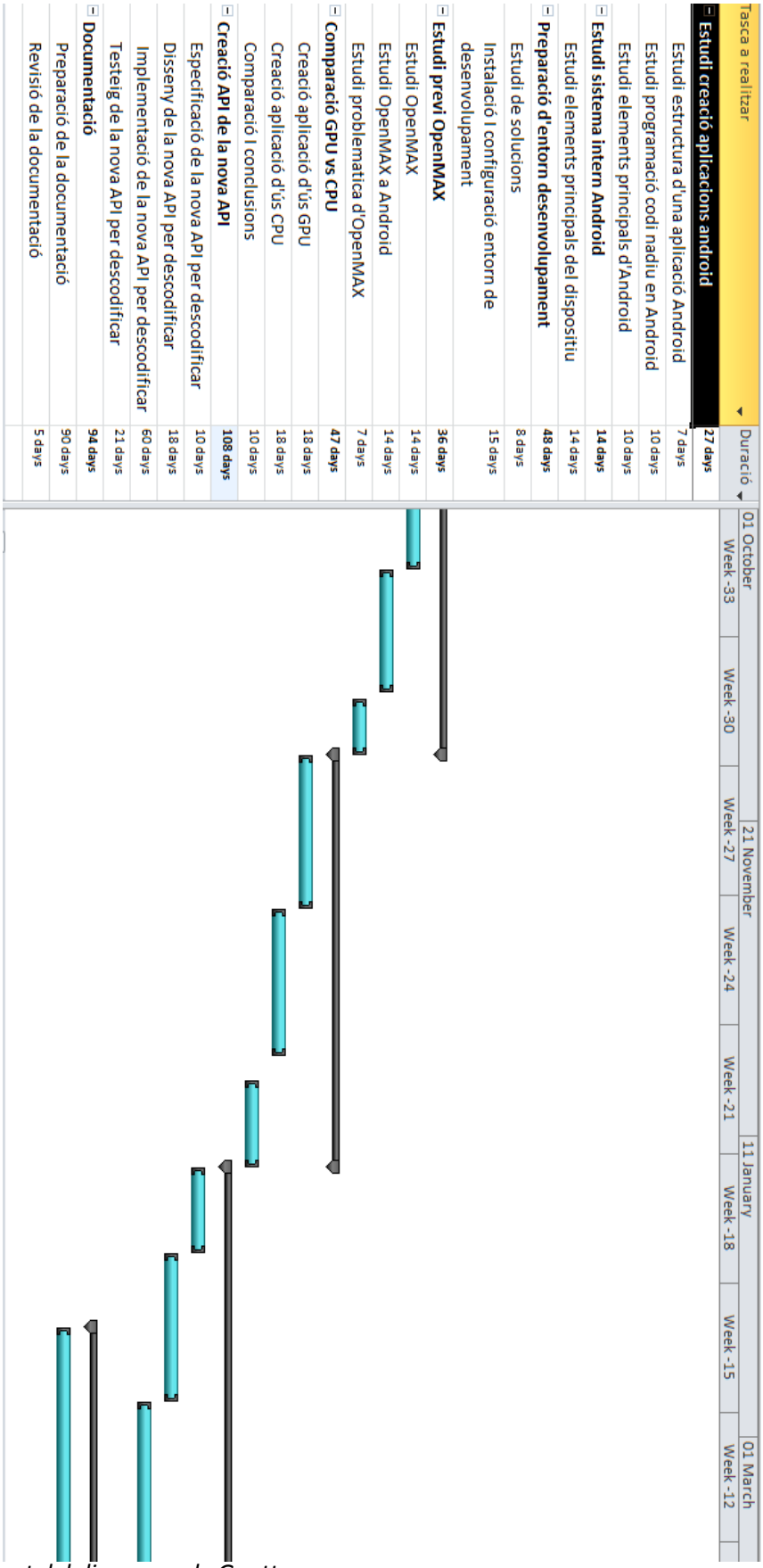

Figura 4: Segona part del diagrama de Gantt

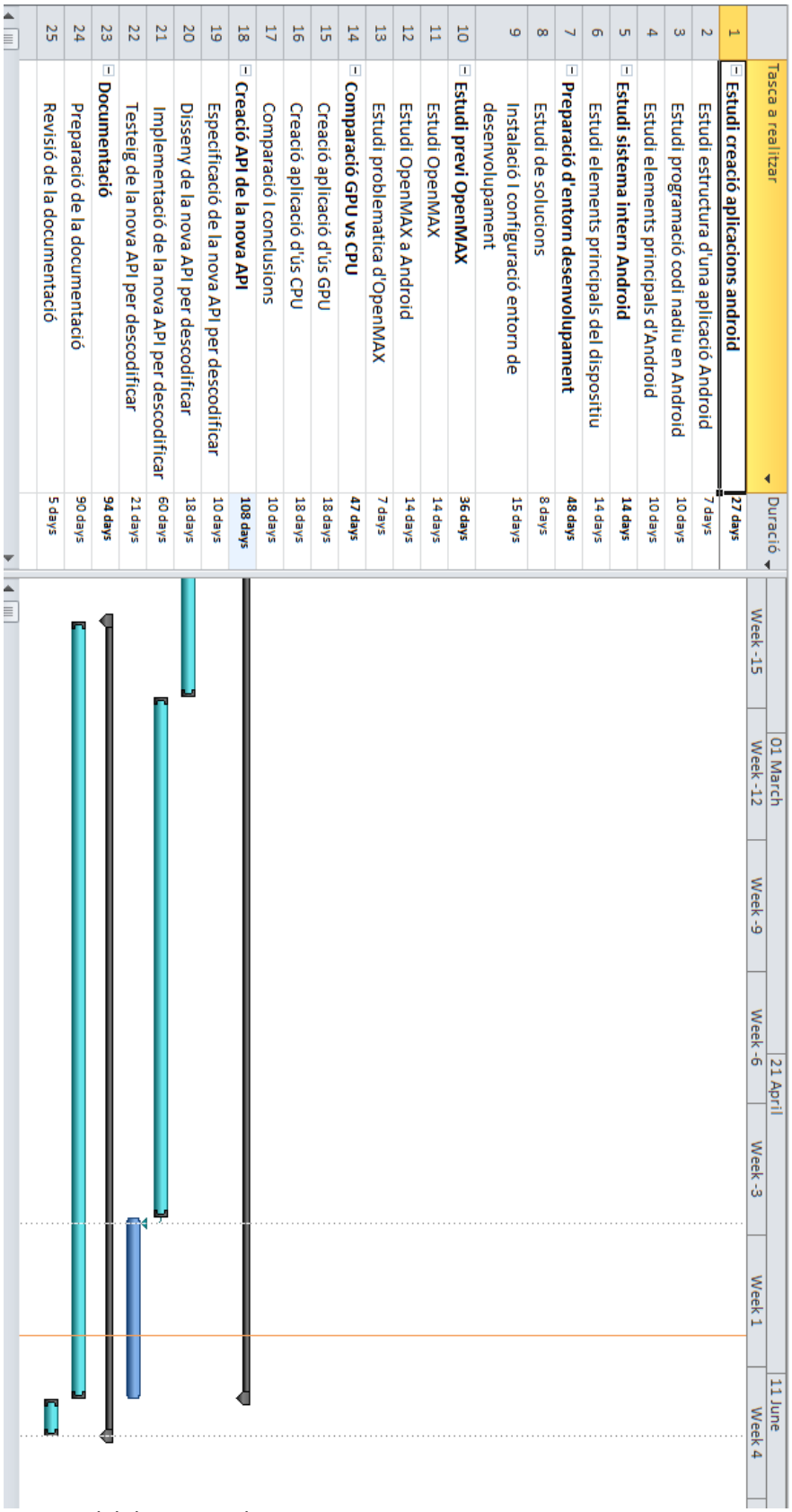

Figura 5: Tercera part del diagrama de Gantt

1.4 Estructuració de la memòria

Amb aquesta secció s'intenta donar una visió estructural de la memòria per a que un lector pugui cercar ràpidament la secció corresponent al seu interès. Per començar cal dir que aquesta memòria ha estat escrita en l'ordre cronològic en el que s'ha anat desenvolupant aquest projecte.

Al capítol 1 s'explica la introducció del projecte; en aquesta secció es dóna una visió global del projecte, els objectius que assolirem, el temps invertit i com ho farem per aconseguir arribar a aquests objectius.

En el capítol 2 es parla de la tecnologia utilitzada; el lector pot fer-se una idea de quin tipus de hardware s'ha necessitat, quin tipus de programes han sigut necessaris per dur a terme aquest projecte i en quines versions d'aquest projecte s'ha treballat.

En el capítol 3 es dóna una visió d'OpenMAX; s'explica en que consisteix OpenMAX i les diferents versions d'aquest. En aquest apartat trobarem tot el necessari per entendre les crides i com es pot arribar a implementar aplicacions utilitzant OpenMAX.

El capítol 4 conté la comprovació de la teoria en que es basa el projecte, és a dir, en aquesta secció s'explica amb detall com s'han construït 2 aplicacions per comprovar el rendiment utilitzant la unitat de processament general (CPU) i d'altre banda la unitat de processament gràfica (GPU).

El capítol 5 conté tot el necessari per entendre com s'ha construït la nostre API de descodificador de vídeo, aquest apartat es divideix en subapartats corresponents a les etapes que han sigut necessàries per crear aquesta API, començant per la fase d'especificació i acabant per la etapa de proves passant per la etapa de disseny i implementació.

En els últims capítols es pot trobar les conclusions obtingudes d'aquest projecte, les possibles ampliacions futures, la bibliografia, el glossari i els agraïments.

2. Tecnologia utilitzada

En aquest capítol mostrarem quina tecnologia s'ha utilitzat per tal de desenvolupar aquest projecte, dividirem aquest capítol en el hardware que s'ha utilitzat i el software que ha sigut necessari i per últim, es farà una breu introducció del sistema operatiu Android i com funciona aquest sistema.

2.1 Hardware

Com aquest projecte està basat en la tecnologia Android, per a realitzar el projecte s'ha triat un smartphone Samsung Galaxy Nexus ja que ofereix la versió d'Android Ice Cream Sandwich 4.0.1, i aquesta versió permet utilitzar les funcions més avençades de l'API de desenvolupament d'Android. Concretament permet l'ús d'OpenMAX AL mitjançant el NDK d'Android que en versions anteriors a la 4.0 no és possible. A més a més, que el fabricant sigui Samsung és un punt a favor, ja que té molts tipus de dispositius al mercat i això ajuda a l'hora de trobar informació per tal de conèixer com funciona el hardware específic d'aquest dispositiu.

Una altra de les raons per la que es va escollir aquest dispositiu és que té un volum raonable d'informació de com és el sistema per dins, tant del hardware com el software. Cosa que en els dispositius més actuals del mercat no hi ha gaire informació encara.

Per últim, al tractar-se d' un dispositiu d'alta gamma permet reproduir vídeos de gran qualitat i amb una resolució força alta per a ser un dispositiu mòbil i això ha permès que es pugui ampliar el ventall de proves per poder treure més conclusions.

Per tal de fer proves de compatibilitat en un sistema anterior de les aplicacions i biblioteques desenvolupades en aquest projecte, s'ha utilitzar un altre dispositiu addicional, el model és un Samsung Galaxy Mini que porta sistema operatiu Android Froyo 2.2.

2.1.1 Samsung Galaxy Nexus

Aquest és el dispositiu utilitzat per provar les aplicacions desenvolupades en aquest projecte. Com ja s'ha dit anteriorment el sistema operatiu que porta instal·lat aquest dispositiu és Android Ice Cream Sandwich 4.0. En la següent fotografia es pot veure el dispositiu Samsung Galaxy Nexus:

Figura 6: Fotografia de Samsung Galaxy Nexus

Característiques tècniques rellevants pel nostre projecte

Aquest smartphone utilitza una CPU del tipus OMAP4460 (dual core a 1,2GHz) i una GPU PowerVR SGX 540 integrades totes dues en un sol xip. El conjunt d'instruccions de la CPU és ARM versió 7. Aquesta CPU ha sigut fabricada pel fabricant de components Texas Instrument.

A aquest dispositiu la GPU (PowerVR SGX540) està inclosa al mateix xip de la CPU (SoC). Aquesta GPU va sortir al mercat al Novembre del 2007 i ofereix píxel, vèrtex i geometria "shader" per hardware i suporta les tecnologies OpenGL 2.0 i DirectX 10.1 i versions anteriors.

Figura 7: Fotografia de la GPU PowerVR SGX540

Texas Instrument dóna la següent taula de característiques a la seva web:

Característiques	**OMAP4460**
Dimensió node	45nm
Freqüència de rellotge	Entre 1.2GHz i 1.5GHz
Rendiment vídeo (2D)	1080p HD 30fps, 1080p HD 48fps
Rendiment vídeo (3D)	720p estereoscòpic 3D, 1080p estereoscòpic 3D
Rendiment d'imatge (2D)	20 MP
Memòria	LPDDR2 400MHz
Rendiment d'imatge (3D)	5MP estèreo (dual), 12MP estèreo (dual)

Taula 4: Característiques del dispositiu Samsung Galaxy Nexus

El dispositiu té un 1GB de memòria RAM. Permet una resolució de pantalla màxima de 1184x720 px, amb un rati de refresc de 60Hz.

Altres característiques no rellevants

Aquest dispositiu té altres característiques no rellevants per aquest projecte com poden ser tecnologia Wifi, GPS, Bluetooth...

2.1.2 Samsung Galaxy Mini

A la següent fotografia es mostra una imatge del dispositiu utilitzat per fer petites proves de compatibilitat amb versions anteriors de les aplicacions desenvolupades en aquets projecte. El sistema operatiu d'aquest dispositiu és Android Froyo 2.3

Figura 8: Fotografia de Samsung Galaxy Mini

Característiques tècniques

La CPU que utilitza aquest dispositiu mòbil és ARMv6 amb una freqüència de rellotge de 600MHz.

Aquest dispositiu té una GPU Adreno 200 (AMD), que permet una resolució de pantalla de 320×200. És compatible amb OpenGL ES 1.1 i 2.0 i té una freqüència de rellotge de 128 MHz

Les altres característiques tècniques són irrellevants en aquest projecte.

2.2 Android

En aquest apartat es donarà una versió simplificada de com funciona el sistema Android operatiu i com s'ha utilitzat per desenvolupar aquest projecte. Android és un sistema operatiu de Google orientat a dispositius mòbils, va ser llançat al octubre del 2008 i està basat en la versió modificada del nucli de Linux 2.6. Android és una plataforma de codi obert distribuïda sota llicència Apache 2.0, per tant la seva distribució és lliure i fa possible l'accés i modificació del codi font.

Com ja s'ha dit a l'Octubre del 2008 es va presentar el primer telèfon G1 (del fabricant HTC), on la seva característica principal va ser el seu sistema operatiu Android. Tot i que aquesta versió estava lluny de les seves gran competidores, la plataforma va començar a créixer exponencialment i actualment és el sistema operatiu més utilitzat (75% de la quota de mercat actual). En la següent figura es mostra un resum de totes les versions existents al mercat:

	Neix Android · Novembre 2007
	G1, primer telèfon Android · Octubre 2008
	SDK Android 1.1 · Febrer 2009
	Android 1.5 Cupcake · Abril 2009
	Android 1.6 Donut · Setembre 2009
	Android 2.0 Eclair · Novembre 2009
	Android 2.2 Froyo · Juny 2010
	Android 2.3 Gingerbread · Desembre 2010
	Android 3.0 Honeycomb · Febrer 2011
	Android 4.0 IceCream Sancdwich · Octubre 2011
	Android 4.1 Jelly Bean · Juny 2012

Taula 5: Versions del sistema operatiu Android

Android IceCream Sandwich 4.0.1

Per aquest projecte s'ha utilitzat la versió d'Android 4.0.1. La estructura del sistema operatiu Android es compon d'aplicacions que s'executen en un framework Java d'aplicacions orientades a objectes sobre el nucli de les biblioteques de Java a una màquina virtual Dalvik amb compilació en temps d'execució.

Els components principals del sistema operatiu Android són: les aplicacions, el framework, les biblioteques, el runtime i el nucli.

Aplicacions

Totes les aplicacions, tant les incloses al propi Android com les desenvolupades, estan escrites en Java. Quan es desenvolupa una aplicació es genera un fitxer de format .apk, aquest fitxer és un paquet autoinstal·lable, això vol dir que conté tots els recursos necessaris per córrer l'aplicació.

Una aplicació pot estar formada per diferents components que poden ser complementaris entre ells o actuar de manera independent. Cal dir que tots aquest components són representats per instancies Java, el que significa que, quan s'executa l'aplicació, es creen instancies d'aquest, a més a més aquestes instancies són independents entre sí. Cadascun dels següents tipus de components tenen una funció associada:

- Activitats: són una simple interfície d'usuari per a la interactivitat amb aquest. En una mateixa aplicació pot haver-hi una activitat per a cada una de les diferents accions que pot realitzar un usuari. Una activitat és una pantalla de la aplicació, al desenvolupament d'aplicacions se la coneix habitualment com a vista.
- Serveis: són uns components que no tenen cap interfície pròpia i que corren en segon pla per a executar tasques d'una llarga durada. Per exemple, una aplicació pot realitzar actualitzacions de la pròpia aplicació en segon pla.
- Proveïdors de contingut: és un tipus de component que permet que es pugui emmagatzemar informació per a que després pugui ser accedida per a varies aplicacions.
- Receptor de transmissions: aquest component crea una notificació en la barra de notificacions del sistema. Aquesta notificació conté informació del que es pretén informar i un apuntador a una activitat o servei que serà executat quan es pressioni sobre aquesta notificació.

Biblioteques

Android inclou un conjunt de biblioteques C. Les biblioteques en codi nadiu del sistema operatiu proveeixen diferents funcionalitats, com poden ser una implementació de la llibreria estàndard de C per Android (anomenada Bionic), un potent i lleuger motor de base de dades relacionals (anomena SQLite), llibreries multimèdies que suportar reproducció i generació d'àudio, vídeo i imatges, una implementació d'OpenGL, un motor per la navegació web (anomenat LibWebCore), una eina per renderitzar mapes de bits i fonts (anomenat FreeType), etc.

Runtime

El runtime conté un conjunt de biblioteques bàsiques que proporcionen una major part de funcions disponibles a les biblioteques del llenguatge Java. Cada aplicació d'Android corre al seu propi procés amb la seva pròpia instància de la màquina virtual Dalvik (d'aquesta màquina virtual es parlarà més endavant).

Nucli

El nucli és la capa més propera al hardware del dispositiu. Aquest nucli (kernel) gestiona la seguretat, la memòria, els processos, la pila i els controladors. El nucli també és l'encarregat del tractament de les interrupcions al sistema.

El nucli serveix de capa d'abstracció entre el hardware i a resta del software. Per realitzar aquesta tasca el software s'executa en diferents modes de seguretat depenent de les necessitats del hardware. Els controladors de dispositius i les parts del nucli s'executen en mode privilegiat, tenint ple accés al hardware sense cap restricció. La resta de software s'executa en mode no privilegiat, tenint accés únicament al hardware a traves del software que s'executa en mode privilegiat (o sigui els controladors). Utilitzant aquesta funcionalitat el nucli actua com a gestor de la memòria proveint al sistema de la seguretat i consistència necessària.

A continuació es pot veure un esquema general de l'arquitectura d'un sistema operatiu Android.

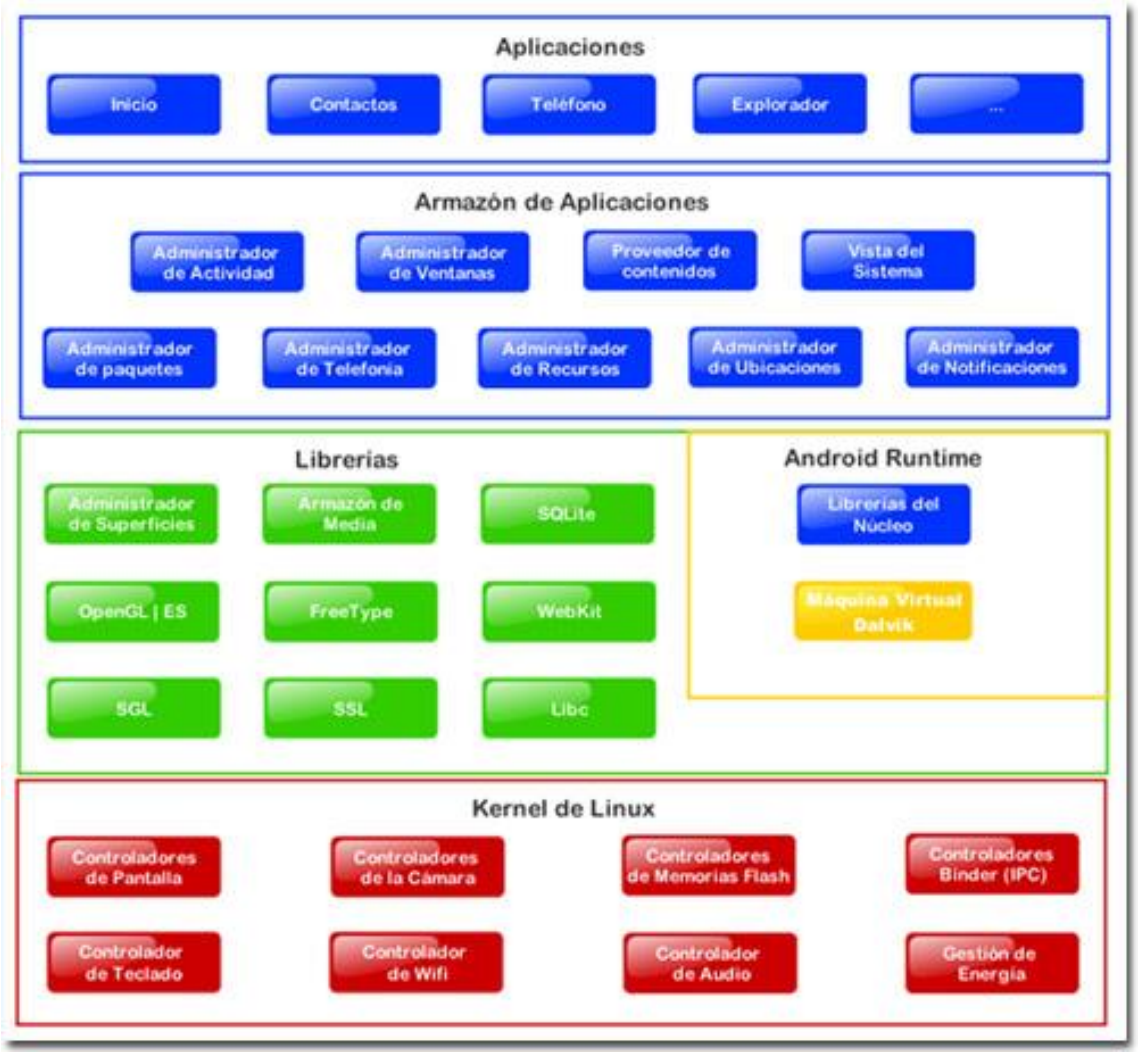

Figura 9: Esquema general d'Android

Dalvik

En Android les aplicacions s'executen amb la seva pròpia instancia mitjançant una màquina virtual. Aquesta màquina virtual és Dalvik. Dalvik és una implementació d'una màquina virtual de Java basada en registres, realitzada per Google i optimitzada per a dispositius intel·ligents. Cada aplicació d'Android executa la seva pròpia instància de Dalvik amb executables .dex, optimitzats per a un millor ús de la memòria. La màquina virtual pot córrer classes compilades en Java transformades prèviament en .dex amb l'eina dx. La decisió de utilitzar aquesta

màquina virtual que en temps real interpreta el codi és la necessitat de que Android sigui un sistema que es pugui utilitzar en diversos dispositius amb una arquitectura diferent. D'aquesta manera evitem compilar el codi per a cada arquitectura diferent i tan sols hem de compilar-lo una única vegada per a que es transformi en codi compatible amb la màquina Dalvik.

També s'ha de tenir en compte la funció del servei Zygote. Aquest servei carrega una instància de Dalvik durant l'inici del sistema i sobre aquesta els diferents tipus de proveïdors de serveis per mitjà del servei del sistema. Quan una aplicació s'inicia es realitza un clon d'aquesta instància i es referencien les zones de memòria assignades als diferents proveïdors de serveis. Si una aplicació requereix escriptura sobre aquestes zones es realitzarà una copia d'aquestes per a la seva utilització local.

2.3 Software

En aquest apartat es pot trobar una descripció de quines aplicacions i programes s'han utilitzat per la creació d'aquest projecte, cal dir que tots els programes que s'han utilitzat han sigut de llicència lliure, per tant s'ha respectat tots els drets de la propietat intel·lectual.

2.3.1 Entorn de desenvolupament

Sistema Operatiu

El projecte s'ha desenvolupat íntegrament en un sistema Ubuntu 12.04 LTS. La decisió d'utilitzar Linux ha sigut per facilitar la connexió amb el dispositiu. Al principi del projecte es va intentar utilitzar el sistema operatiu Windows amb resultats poc exitosos, ja que per tal de poder programar en codi natiu a Windows es necessari una aplicació anomenada Cygwin que donava complexitat innecessària al projecte, per tant es va decidir continuar amb el Sistema Operatiu Linux. El processador d'aquest ordinador en que s'ha desenvolupat el projecte és un Pentium 4 a 1,73 GHz de 32 bits.

Per fer la documentació s'ha utilitzat un ordinador amb un sistema Windows 7 amb Microsoft Word 2010.

Java

La programació de les vistes i activitats de les aplicacions està feta en Java ja que és el llenguatge de programació d'alt nivell orientat a objectes que s'utilitza per crear aplicacions per Android. De fet, per ser sincers, Android utilitza un subconjunt de Java. En aquest projecte s'ha utilitzat la versió 6 de Java.

Eclipse

Eclipse és un IDE (Integrated Development Environment) amb llicència lliure, que ens permet programar per Android d'una manera fàcil i senzilla. També permet crear interfícies gràfiques, afegir paquets, depurar aplicacions i exportar aplicacions de manera senzilla i eficaç.

Al Eclipse s'ha hagut de configurar el plugin d'Android per tal de aconseguir que el SDK d'Android s'integri amb l'Eclipse. També s'han instal·lat els plugins i llibreries necessàries per poder programar en java i en C / C++.

La versió d'Eclipse utilitzada en aquest projecte és Eclipse Indigo 3.7.2.

SDK Android

Per tal de desenvolupar aplicacions en Android s'ha d'utilitzar un kit de desenvolupament desenvolupat per Google. En aquest projecte s'ha utilitzat l'ultima versió existent al mercat del Software Development Kit (SDK) per sistemes Linux, concretament s'ha utilitzat la versió 20. Aquest Kit conté tot el necessari per desenvolupar aplicacions i executar-les a l'entorn Android. Aquest kit compta amb una aplicació anomenada adb (Android Debug Bridge), aquesta aplicació permet la comunicació per USB amb el dispositiu mòbil, permetent accedir al Shell del dispositiu mitjançant el terminal del ordinador.

També el kit inclou documentació, llibreries, exemples i imatges per les diferents versions d'Android. Les últimes versions també inclouen un emulador per tal de debuggar aplicacions. No s'ha pogut utilitzar l'emulador perquè aquest no inclou la GPU. Tot i que a l'última versió si que s'inclou, l'emulador de GPU està en fase beta i donava molts problemes de compatibilitat, per tant s'ha preferit testejar directament les aplicacions i biblioteques al dispositiu.

2.3.2 Programació codi nadiu

Per a construir i compilar el codi per una arquitectura ARM des d'una altre arquitectura (en el nostre cas serà Linux) s'ha d'utilitzar un compilador creuat. Aquest compilador és molt útil perquè compilar directament al dispositiu és molt incòmode i normalment no tenim els privilegis necessaris per fer-ho. En canvi amb el compilar creuat, es compila des de el PC i després la instal·les al dispositiu, fent el mètode molt més senzill.

L'API de desenvolupament de codi nadiu d'Android (NDK) té un conjunt de binaris per realitzar compilacions creuades per arquitectures ARM des d'un sistema Linux. Només se li ha d'indicar quin és el dispositiu final (--target XX) on XX és la versió del ARM del dispositiu mòbil.

NDK Android

NDK (Native Development Kit) és útil per poder programar en codi natiu, aquest genera un paquet autoinstal·lable (.apk) que el dispositiu reconeix i l'instal·la. En aquest projecte s'utilitza el NDK per desenvolupar la aplicació en codi C (recordem que el llenguatge C és l'utilitzat per OpenMax). La versió que s'ha utilitzat en aquest projecte és la 8b.

Característiques del codi natiu

Primer de tot cal dir que programar en codi natiu sempre serà més complex i pot ser no serà més eficient, és a dir programar en codi natiu no és garantia d'èxit. En aquest projecte és necessari programar en codi natiu perquè s'ha de programar en C per tal de poder utilitzar OpenMAX, OpenMAX actualment no té cap suport per a llenguatges Java.

Cal remarcar també que es pot desenvolupar tota la aplicació en codi natiu (inclòs les activitats) però no es recomanable. Ja que és molt més senzill desenvolupar les activitats en Java.

La millor forma de treballar és crear una funció C / C++ que faci tot el procediment costós, i cridar-la des de Java mitjançant el JNI (Java Native Interface). En el nostre cas s'han fet les activitats, el reproductor i totes les funcions de control a Java i després la connexió amb OpenMax és fa amb codi nadiu.

Una nota important a nivell de seguretat és que si s'utilitza el compilador creuat per desenvolupar aplicacions, les aplicacions creades estaran subjectes a les mateixes condicions de seguretat que les aplicacions creades per a córrer en la maquina virtual d'Android.

Una altre característica important és que el codi nadiu no ofereix un recol·lector de brossa, és a dir s'ha d'alliberar els punters explícitament. També és important recordar que al desenvolupar una aplicació en codi nadiu aquesta perd la part més important del llenguatge Java, la seva portabilitat.

Per crear una aplicació en codi nadiu, primer de tot s'ha de crear un directori anomenat JNI (Java Native Interface). Aquest directori és on es fa el mapeig de Java amb el codi nadiu i és on s'escriurà totes les crides en codi C o C++. També s'ha de crear la capçaleres per les funcions

escrites en C. És important modificar el nostre fitxer Android.mk per tal d'incloure les biblioteques natives i els fitxers escrits en C.

Eclipse té una funció molt útil on es pot convertir un projecte escrit en Java en un projecte mixt en Java i C. Tot i que no es necessari, és molt útil a l' hora de depurar el codi.

2.3.3 Problemàtica Android

S'ha comentat alguns dels avantatges de desenvolupar aplicacions per a un sistema obert com és Android. Però realment el fet que sigui obert és el seu principal desavantatge. El problema radica en que Android es va actualitzant periòdicament, però com és un sistema operatiu que s'implementa en molts dispositius, l' actualització d'aquest a l'última versió del sistema depèn dels fabricants.

La cronologia és la següent: Google allibera una nova versió d'Android i els fabricants de cada dispositiu tenen que adaptar els seus drivers a aquesta versió, i si o desitgen, desenvolupar una interfície visual pel nou sistema operatiu. Seguidament abans d' actualitzar els terminals, si aquest no són lliures, les actualitzacions passen a mans de les companyies operadores de telefonia mòbil, les quals s'encarreguen d'afegir una sèrie d'aplicacions que després l'usuari no pot esborrar. Tot aquest procés implica que les actualitzacions arribin al dispositius molt tard. Però aquest no és el major problema, ja que molts terminals no s'arriben a actualitzar, perquè als fabricants no els interessa actualitzar els dispositiu que porten cert temps al mercat.

Tot això provoca que de cara al desenvolupador d'aplicacions es presenti una fragmentació entre versions bastant complexa. Una aplicació desenvolupada per a una versió serà compatible amb totes les futures versions, però no és així al contrari. Degut a que Android va evolucionant i va incloent noves operacions que faciliten les coses i que no estan suportades per versions anteriors a aquestes.

2.3.4 Format de vídeo: H264

En aquest aparta s'explicarà en que consisteix el protocol de vídeo H264, recordem que aquest protocol és el que s'ha utilitzat en aquest projecte.

Descripció

H264 és un còdec digital de compressió estàndard. El sistema el va dissenyar la UIT (Unió internacional de telecomunicacions) a Ginebra l'any 2003. La principal funció del H264, també conegut com AVC, és sense incrementar la complexitat del disseny dels anteriors estàndards (H263, MPEG-2...) ser capaç de transportar el vídeo amb la mateixa qualitat d'imatge i menors bit-rates. És a dir amb menys consum d'ample de banda i menys pes d'arxiu, la mateixa qualitat.

Compressió de vídeo

La compressió de vídeo es basa en la reducció de redundàncies temporals i redundàncies espacials, les redundàncies temporals són aquelles similituds entre imatges consecutives. Es pot estalviar una gran quantitat d'ampla de banda i memòria si nomes s'envien els blocs que canvien d'imatge. Bàsicament l'H264 intenta aprofitar les redundàncies espacials i les dependències temporals existent entre els diferents frames. El resultat és un vídeo en streaming que compren un frame de referència (anomenat I Frame) i les àrees que canvien de la imatge (s'anomenen P o B frames) que es van envien i cada cert temps es torna a enviar el frame de referencia (I frame)

El número de frames de distància entre un I frame i el següent en el streaming és diu distància GOV. Aquest paràmetre es pot canviar i per tant a més distància, menys I frames així que la tassa de bits serà menor. Però si augmentem molt la distancia GOV hi ha un risc que la imatge pugui ser inconsistent.

En la següent figura es mostra només els frames que canvien de color vermell. És a dir, en aquest cas s'envia primer un frame amb tota l'escena i després s'envien els frames de color vermell que són els canvis (com es pot apreciar a la segona imatge). D'aquesta manera s'estalvia enviar tota la escena sencera.

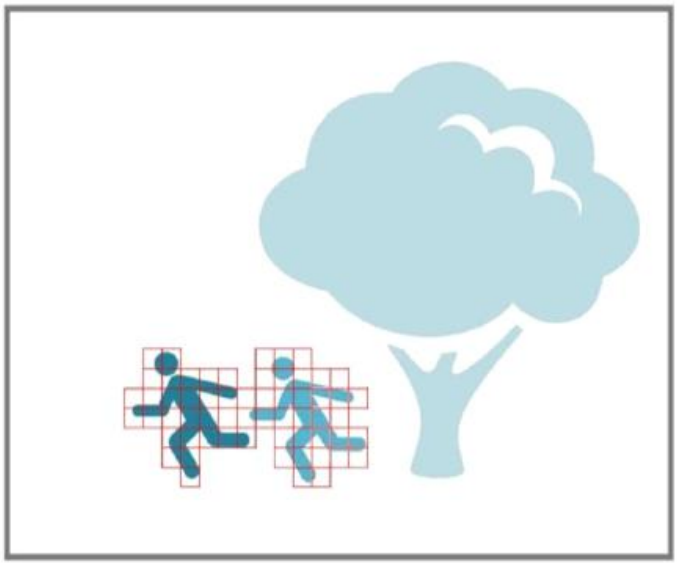

Figura 10: Exemple d'imatge on només canvien de posició les persones

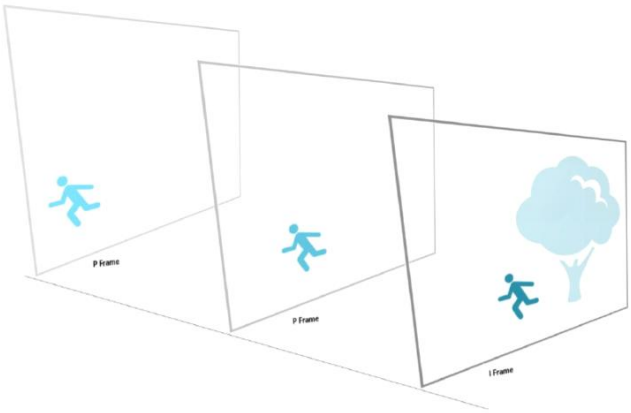

Figura 11: Exemple d'imatge on només canvien de posició les persones

Compressió de vídeo avançada H264

H264 és una evolució del H263 i per tant suporta més característiques avançades com les descrites a continuació.

Una de les característiques que suporta l'H264 és el Motion Compensation. En una frame P o B, només es transmet el bloc que canvia i el Motion Compensation permet transmetre només el vector de moviment. El descodificador permet moure el bloc, que ja ha sigut prèviament retransmet, a la nova localització fent servir el vector de moviment. Això estalvia una quantitat significant de tassa de bits comparat amb l'anterior sistema.

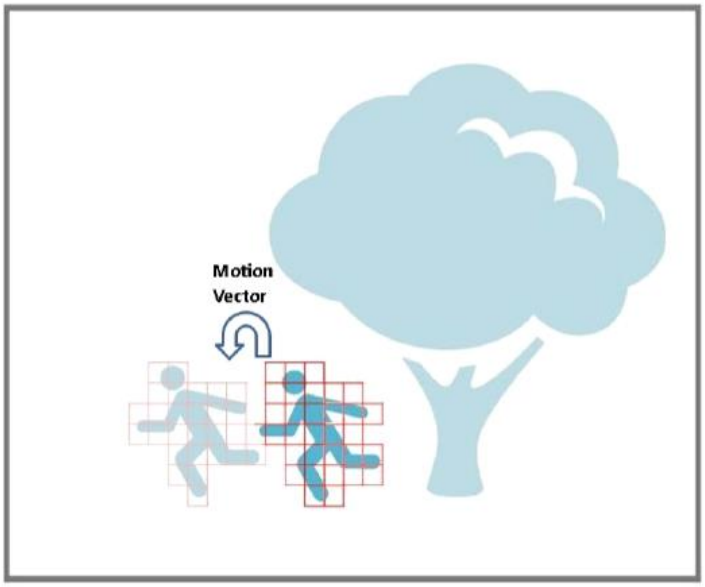

Figura 12: Exemple d'imatge amb el vector Motion

Una altre característica que suporta l'H264 és l'anomenat "deblocking filter". Amb això s'aconsegueix una millora en la qualitat d'imatge, però també permet a l'usuari configurar una compressió del vídeo més alta. Aquesta tècnica consisteix en fer "smoothing" (allisar) els marges entre els blocs que permet guanyar qualitat del vídeo. Com per fer la compressió de la imatge, els colors de cada bloc es converteixen en més estàndards, és a dir, igualant el color dels píxels que estan a prop. Si aquest efecte creix, els marges dels blocs es poden fer més visibles provocant l'efecte pixelació. Com la següent imatge:

Figura 13: Exemple d'imatge aplicant smoothing

A la primera imatge es veu un elevat grau de pixelació, a la segona aplicant el filtre es pot veure que s'han suavitzat bastant els marges.

El "deblocking filter" impedeix que això passi mitjançant uns píxels tipus en cada costat del marge del bloc. Basat en aquest píxels el filtre decideix el color mig i tornar a pintar el marge de cada costat del bloc, això crea l'efecte "smoother" entre blocs.

Estructura

El descodificador treballa amb una seqüència de bits rebuts en un format específic. Aquesta seqüència esta dividida en paquets. Com es pot veure en la següent imatge el fitxer esta dividit en NAL-paquets amb la següent forma:

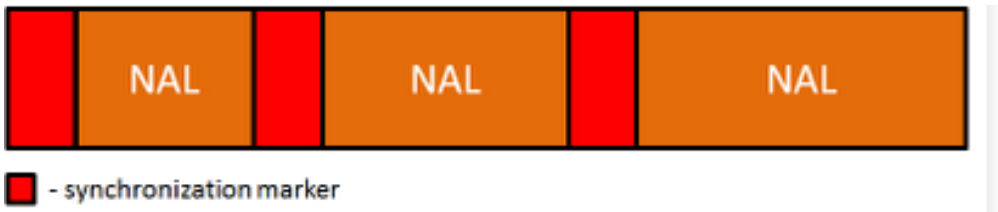

Figura 14: Divisió paquets H264

El primer byte del NAL-paquet (Network Abstraction Layer) és la capçalera que conté informació sobre el tipus d'estructura del paquet. Tots els possibles paquets estan descrits a la següent taula:

Tipus	Definició
0	Indefinit
1	Fragment de capa sense partició no-IDR
2	Fragment de dades nivell A de partició
3	Fragment de dades nivell B de partició
4	Fragment de dades nivell C de partició
5	Fragment de capa sense partició IDR

6	Informació addicional (SEI)
7	Configuració del paràmetre de seqüència
8	Configuració del paràmetre de la imatge
9	Delimitació de la unitat d'accés
10	Final de la seqüència
11	Final del fitxer
12	Dades
13..23	Reservat
24..31	Indefinida

Taula 6: Tipus de paquets H264

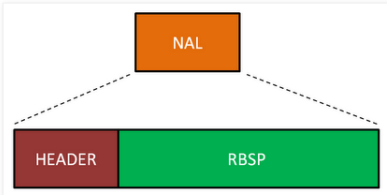

Figura 15: Descomposició de paquet H264

Com es pot veure a la figura anterior, el paquet s'identifica com RBSP (Raw Byte Sequence Payload). RBSP descriu una tira de bits en un específic ordre de SODB (Cadena de bits de dades). És a dir, RBSP conté SODB (per exemple si SODB és buit també ho estarà el RBSP).

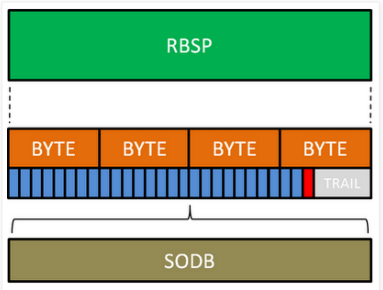

Figura 16: Descomposició de paquet RBSP a H264

Com es pot apreciar a la següent figura cada “slice” està dividit en macroblocs. Normalment cada imatge codificada correspon a un “slice” però també una imatge pot tenir molts “slice”

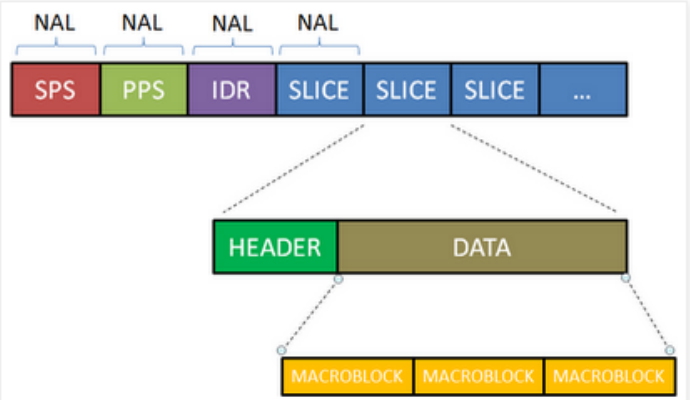

*Figura 17: Mínima d*escomposició de paquet a H264

Tipus	**Descripció**
0	P-slice. Consisteix en P-macroblocs o I-macroblocs
1	B-slice. Consisteix enBP-macroblocs o I-macroblocs
2	I-slice. Consisteix en I-macroblocs. Cada macrobloc és una previsió de blocs anteriors
3	SP-slice. Consisteix en P i I-macrobloc i permet canviar entre fitxers codificats
4	SI-slice. Consisteix en un tipus de SI-macroblocs i permet canviar fitxers codificats
5	P-slice.
6	B-slice.
7	I-slice.
8	SP-slice.
9	SI-slice.

Taula 7: Tipus de macroblocs a H264

Els macroblocs són els que contenen més informació, com la lluminositat i el cromat corresponent a píxels individuals. A continuació podem veure com és un macrobloc per dins.

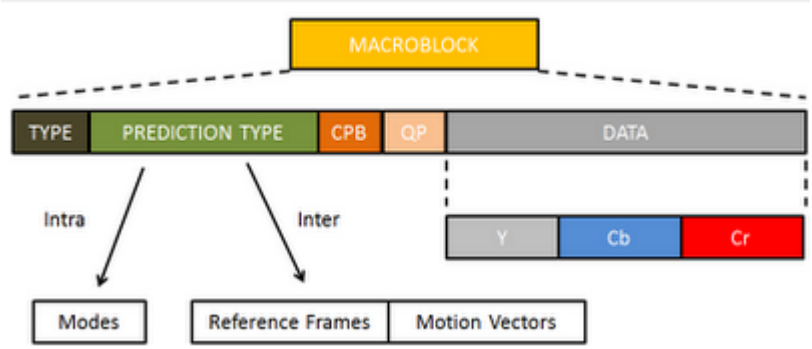

*Figura 18: Mínima d*escomposició de macrobloc a H264

3. OpenMAX

A aquest capítol es donarà una visió global de que és OpenMAX i de quins nivells d'arquitectura hi ha actualment a OpenMAX, també s'explicaran quines diferents versions hi ha d'OpenMAX a Android i les diferents tipus d'implementacions. Per últim, explicarem la problemàtica que hi ha actualment amb OpenMAX.

Figura 19: Logotip d'OpenMAX

Figura 20: Logotip de Khronos Group

3.1 Introducció

OpenMAX és una API (Interfície de Programació d'Aplicacions) multiplataforma, de llicència lliure i escrita en C que dóna una abstracció per rutines especialment útils per àudio, vídeo i imatges. El principal objectiu és reduir el cost i la complexitat de la portabilitat al software multimèdia pels diferents processadors i arquitectures.

OpenMAX ha estat dissenyat i implementat pel Grup Khronos. El grup Khronos va ser fundat al Gener del 2000 per un gran grup de companyies líders en el sector de multimèdia. Entre elles es troben 3Dlabs, ATI, Discreet, Evans & Sutherland, Intel, NVIDIA, SGI i Sun Microsystems. L'objectiu d'aquestes companyies era clar, crear una API que fos estàndard amb les màximes funcionalitats multimèdia possibles i que sigui compatible a una gran varietat de plataformes i dispositius diferents.

OpenMAX té 3 capes d' interfícies que treballen en diferents nivells. Les capes són les següents:

- OpenMAX AL: Capa d'aplicació és una interfície a alt nivell per al desenvolupadors de software multimèdia.

- OpenMAX IL: Capa d'integració és el middleware que permet la comunicació entre components i estandarditzada entre diferents plataformes.

- OpenMAX DL: Capa de desenvolupament és una interfície a baix nivell que permet a les companyies integrar el software existent fàcilment amb el nou hardware.

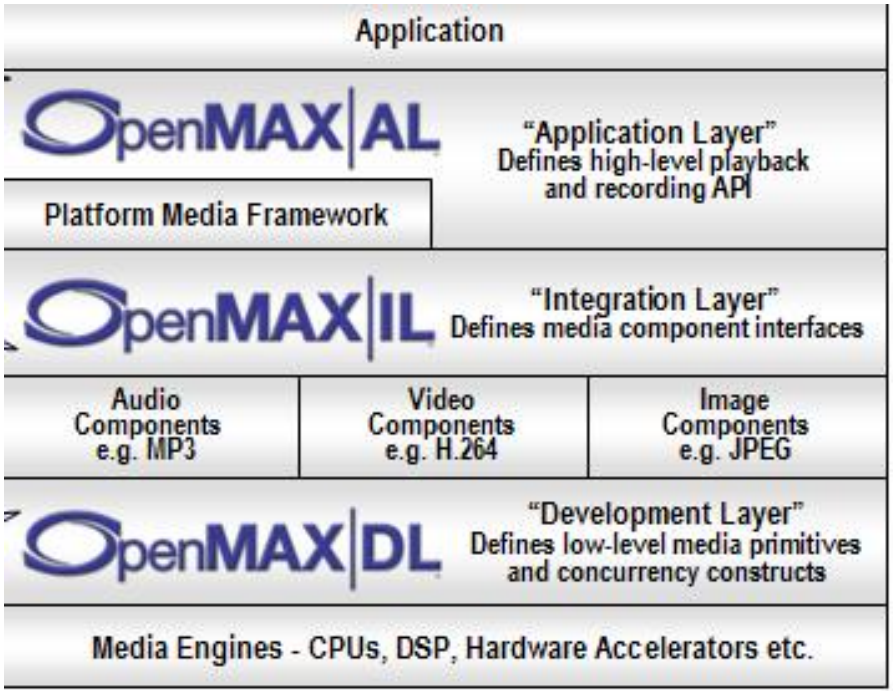

Figura 21: Arquitectura en capes d'OpenMAX

3.2 Aplication Level (AL)

La capa més externa és OpenMAX AL (Aplicattion layer). Aquesta capa dóna una interfície estàndard entre l' aplicació i el middleware encarregat de la part multimèdia. Mentre el middleware dóna el serveis necessaris per aconseguir totes les funcionalitats desitjades, OpenMAX AL dóna la portabilitat necessària.

Existeixen 3 versions d'OpenMAX AL. La primera va ser publicada al Juny del 2009 i és la versió 1.0, al març del 2010 va aparèixer la versió 1.0.1 (aquesta versió és la implementada en Android) i al Gener del 2011 es va desenvolupar la versió actual 1.1

OpenMAX AL funciona independentment del dispositiu físic i és multiplataforma. OpenMAX AL també pot ser aplicat per a dispositius més sofisticats o específics de comunicació, de reproducció o de gravació.

Aquesta API funciona de la següent manera. OpenMAX AL organitza l' abstracció de les funcions al voltant d'un conjunt d' objectes. Una aplicació adquireix tots els objectes d'un objecte motor, el qual encapsula una sessió d'OpenMAX i serveis de tots els objectes AL. El principal objecte d'aquest s'anomena "media object". Aquest objecte representa tant la tasca de reproduir (media player) com la de gravar (media recorder), que són les que s'encarreguen d'agafar les dades d'una respectiva font i enviar-les a un destí designat.

Característiques

- Reproductor de medià: Inclou reproductor d'àudio, codificador d'àudio i de so, reproductor de vídeo i imatges, codificador de vídeo i imatges.

- Gravador de vídeo: Inclou suport per gravar vídeo i àudio i per capturar imatges.

- Efectes i controls: Inclou controls com per exemple el control del volum i el balanç. A més a més permet efectes de la música com el equalitzador. Per vídeo inclou ajustaments com la lluentor i el contrast.

- MIDI: format per els tons de mòbils. També inclou SP-MIDI, mobile DLS, mobile XMF.

- Radio analògica: format RDS/RBDS.

- Display de LEDS: inclou suport per LED de colors.

- Dispositius vibratoris: inclous suport per controlar la intensitat i la freqüència d'aquest.

Integració amb OpenSL

OpenMAX és pot utilitzar com a complement d'OpenSL. OpenSL ES és una altra API del grup Khronos que es complementa amb el OpenMAX. Tots 2 tenen la mateixa arquitectura, però OpenSL esta especialitzat en les biblioteques de so. Tots dos comparteixen algunes funcionalitats en comú, però OpenSL ES dóna suport a avançades funcionalitats pel que fa a l'àudio, com pot ser el so estèreo.

3.3 Integration Level (IL)

En aquest apartat es farà una breu descripció de la capa d'OpenMAX IL, cal dir que la descripció en detall d'aquesta capa es farà al capítol d' implementació de l'API de descodificació de vídeo.

Descripció

OpenMAX IL (Integration Layer) és la següent capa d'OpenMAX. Aquesta capa, com la resta, està escrita en llenguatge C i s'encarrega de donar portabilitat a les diferents plataformes resultant totalment transparent a l'usuari. Per exemple, OpenMAX IL permet a l'usuari carregar, controlar, connectar i descarregar components.

La principal funcionalitat de OpenMAX IL és donar una abstracció del hardware i del software de l'arquitectura del sistema, per solucionar el problema de portabilitat entre la gran quantitat de sistemes. Sense aquesta interfície, els venedors haurien de escriure un codi per cada un dels dispositius que hi ha al mercat.

Sovint a la realitat, en molt sistemes ja existeix un framework a nivell d'usuari. Per tant OpenMAX IL esta dissenyat per integrar-se a sota d'aquest sense produir cap tipus de treball doble, ni overhead entre els 2 frameworks. Addicionalment OpenMAX IL pot ser més còmodament integrat si OpenMAX AL està implementat en el sistema. La següent figura mostra l'escenari ideal.

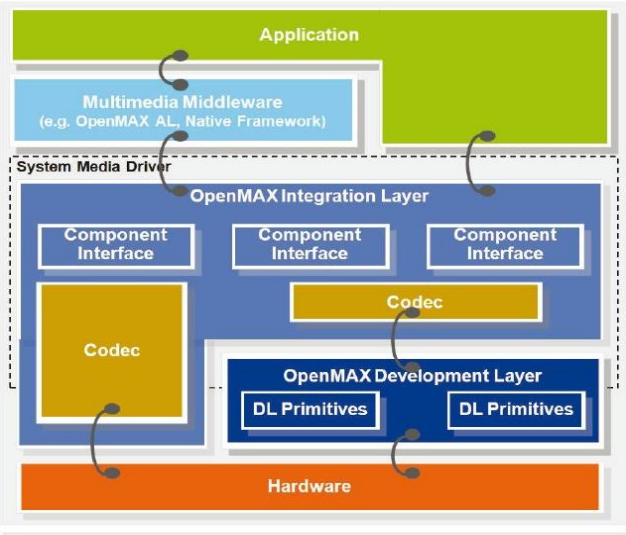

Figura 22: Arquitectura d'OpenMAX IL

Dominis

OpenMAX IL esta dividit en 3 dominis:

- Àudio

- Vídeo

- Imatge

També dóna suport per altres dominis, com per exemple el rellotge o combinacions de dominis, com pot ser el processador de vídeo i imatge.

Implementació

OpenMAX IL esta basat en 2 segments: el nucli (core) i els components.

El nucli s'encarrega de carregar i descarregar components i facilitar la comunicació entre ells. Una vegada carregat, l'API permet a l'usuari comunicar-se directament amb el component, amb l'avantatge que elimina l'overhead de totes les comandes addicionals. Una vegada establerta la comunicació, el nucli ja no s'utilitza i les comunicacions entre components funcionen directament.

A més a més el nucli dóna la llista de components i les funcionalitats que suporten aquestes.

D' altre banda, cada component representa una funcionalitat, com per exemple, còdecs, filtres, mescladors... Depenen de la implementació, un component pot representar una part de hardware, del software o una combinació d'aquest. L'estat del buffer, els error o altres tipus d'informació s'envien a la aplicació mitjançant un conjunt de funcions.

La comunicació entre components es fa mitjançant ports. Els ports s'encarreguen de fer la connexió entre els components mitjançant data streaming (port d'entrada i de sortida) i també s'encarreguen dels buffers que es necessiten per mantenir la connexió. Els ports es poden desactivar i activar independentment sense afectar el funcionament del component.

3.4 Development Level (DL)

OpenMAX DL (Development Layer) és l'última capa i treballa a baix nivell amb el firmware i el hardware . Aquesta capa defineix un conjunt de funcions i instruccions de baix nivell. OpenMAX DL permet portabilitat entre diferents varietats d'arquitectures hardware com poden ser les CPUs, les GPUs o els DSP. El principal objectiu de OpenMAX DL és habilitar la portabilitat entre els chips i els còdecs, això ho aconsegueix amb un conjunt de funcions primitives.

Dominis

OpenMAX DL esta dividit en 5 dominis:

- AC: Còdecs d'àudio (MP3 i ACC)
- IC: Còdecs d'imatge (JPEG)
- IP: Processar imatges (funcions genèriques per imatges)
- SP: Processar senyals (funcions genèriques per àudio)
- VC: Còdecs de vídeo (H264 i MP4)

Cada domini està organitzat en diferents col·leccions de domini anomenades, omxAC, omxIC, omxIP, omxSP i omxVC. Cada domini està descompost en diferents sub-dominis que a la vegada estan especialitzats en diferents multimèdia còdecs.

Implementacions

OpenMAX DL té diferents tipus d'implementacions que permeten una portabilitat de manera eficient entre les diferents plataformes. Les diferents metodologies d'implementacions són de manera síncrona, de manera asíncrona i la integrada (sistemes concurrents). S'escull un tipus d'implementació depenen del tipus de hardware.

3.5 Implementacions d'OpenMAX

En aquest apartat farem un breu repàs de les implementacions actuals d'OpenMAX.

Bellagio (OpenMAX IL)

Bellagio és una implementació de OpenMAX IL dissenyat especialment per funcionar en Linux. Facilita que els desenvolupadors de Linux puguin desenvolupar aplicacions de multimèdia i streaming en diferents components. Aquesta implementació és exclusivament per Linux d'arquitectura x86 i plataformes ARM.

G Streamer

GStreamer és un framework multimèdia amb llicència GNU i multiplataforma que permet crear aplicacions audiovisuals, vídeo, so o combinacions d'aquestes. Aquesta implementació suporta diferents sistemes operatius, processadors i compiladors.

LIM OpenMAX IL i OpenMAX AL

Possiblement la implementació més famosa d'OpenMAX. Va ser creada al 2011 amb llicència lliure. La nova versió permet la càrrega dinàmica i estàtica de diferents dispositius, tunneling, buffer sharing i core shadow entre altres. Versions anteriors donen suport a quasi totes les funcionalitats d'OpenMAX, tant la capa d'aplicació com la d'integració.

DaVinci

És una implementació de lectura de vídeo MPEG-4 H.264/AVC utilitza OpenMAX AL i OpenMAX IL estàndard. Aquesta implementació esta feta en el sistema operatiu Linux.

3.6 OpenMAX a Android

A aquest punt es dóna una breu descripció de com OpenMAX evoluciona a Android, i on es pot trobar OpenMAX dintre de cada versió d'Android.

OpenMAX a Android es pot dividir en 3 grans grups depenent de les versions del sistema operatiu. Cal remarcar que tot i que OpenMAX vol aconseguir estandarditzar el procés hi ha diferències significants entre les versions 2 i 4 d'Android.

A part dels diferents tipus d'OpenMAX als dispositius, també depenen les diferents versions d'aquest, i a més a més hi ha diferències a l' àmbit dels fabricants, ja que cada fabricant té els seus propis còdecs.

OpenMAX a Android 2.XX

A Android 2 només es pot accedir a OpenMAX mitjançant la capa IL, accedir mitjançant la capa IL no es gens trivial com podrem comprovar més endavant. Aquestes versions d'Android no tenen cap API per tal d'accedir a la capa AL. Per tal de poder utilitzar-lo s'ha de tenir grans coneixements de com funciona OpenMAX i de com es mapeja aquestes crides a l'entorn Android.

La versió 2.1 és la primera versió que suporta l'acceleració de hardware en format MPEG.

OpenMAX a Android 3.XX

Aquesta és una versió d'Android poc comercial, pocs dispositius utilitzen 3.0 (només unes poques tabletes del mercat). Tot i que aquesta versió no porta grans millores, és la primera versió que suporta l'acceleració de hardware en format H264.

OpenMAX a Android 4.XX

Aquesta versió Android comença a donar més medis per tal d'accedir a les capes inferiors d'OpenMAX sense haver d'utilitzar la OpenMAX IL. A la versió 4.0 és la primera que dóna suport per tal d'accedir als còdecs de vídeo a nivell de Java. Per fer-ho s'ha de escriure en codi nadiu que interactua directament amb OpenMAX.

A més Android 4.0 dona suport a OpenMAX AL 1.0.1 com a part del seu NDK (Native Development Kit). Aquest NDK permet desenvolupar software directament sobre una

plataforma i no sobre una màquina virtual. A més el NDK ofereix exemples i documentació sobre OpenMAX AL a Android.

Aquesta implementació dona una interfície en llenguatge C (també es pot cridar des del llenguatge C++) que es pot utilitzar en dispositius 4.0 o superiors.

Cal dir que aquesta implementació d'OpenMAX AL te funcionalitats limitades. L'àudio no pot ser configurat i no es pot aplicar cap efecte de so. A més a més, no suporta directament un accés a la càmera del dispositiu.

A la versió Android 4.1 proveeix accés a la acceleració de hardware (GPU) al nivell d'aplicació a traves d'una API Java. Tot i que aquesta API té certes limitacions en quan a configuracions. A més aquesta API Java no permet la codificació de vídeo i està subjecta al format H264 amb transport .ts.

3.7 Problemàtica actual

Una vegada vist tot l'estudi previ d'OpenMAX i d'Android es pot treure conclusions sobre els problemes que hi ha a l'hora d'utilitzar OpenMAX per poder accedir al hardware en dispositius Android.

OpenMAX és massa complexa, sense tenir en compte els 3 tipus de capes que disposa, si ens centrem en una capa com pot ser IL, hem de tenir em compte 6 estats, la configuració dels ports, la configuració dels components i, a més a més, això pot ser específic de cada protocol.

La documentació que es pot trobar d'OpenMAX són manuals de més de 500 pàgines per cada nivell d'arquitectura. Per entendre aquest manuals es requereix un gran nivell de coneixements informàtics i un domini acceptable d'anglès ja que es troben exclusivament en anglès. L'única manera de entendre com funciona realment OpenMAX és mitjançant un procés de enginyeria inversa, ja que hi ha diferents versions d'OpenMAX.

Android no dóna cap tipus de suport al format H264, per tant per desenvolupar una aplicació s'ha de fer servir OpenMAX, ja sigui directament o indirectament (mitjançant una llibreria que utilitza OpenMAX).

Es pot trobar diferents versions d'OpenMAX depenent del fabricant del dispositiu, fins i tot, es pot trobar diferents versions al mateix fabricant i model. Per posar un exemple amb el mateix codi font és impossible obtenir els mateixos resultats dintre dels diferents dispositius de la gamma Nexus de mòbils fabricats per Google.

Degut a aquests problemes surt la idea del nostre projecte i és que el desenvolupador d'aplicacions pugui crear les aplicacions de manera senzilla, és a dir, sense tenir en consideració el fabricant del dispositiu, les versions d'OpenMAX existents, les versions d'Android i a més a més sense la complexitat d'OpenMAX.

4. Estudi previ: Comparació GPU i CPU

Abans de començar a dissenyar i desenvolupar la nova API per tal de descodificar vídeo utilitzant la GPU, s'ha de tenir en compte com es pot arribar a aconseguir aquest objectiu. S'ha de demostrar que utilitzant la GPU del dispositiu s'aconsegueix un guany respecte a no utilitzar-la. Per tant, l'estudi previ es basa en contestar la pregunta: quin guany s'aconsegueix utilitzant la GPU? és realment un guany? Per contestar aquesta pregunta s'han de desenvolupar diverses aplicacions per tal de poder comparar-les entre elles.

Sense entrar en detalls sobre la tecnologia utilitzada, la següent figura ens mostra quins components tindrà la nostra aplicació. Per fer aquesta aproximació de funcionalitats s'ha pres com a referència la classe estàndard MediaPlayer d'Android.

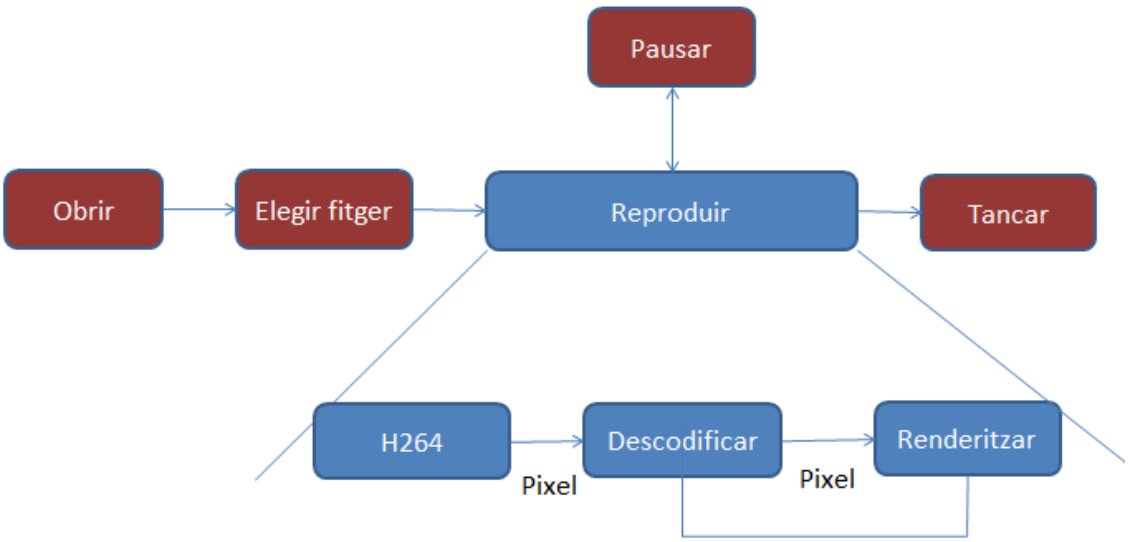

Figura 23: Funcionalitats de la nostra aplicació

O sigui s'ha d'obrir la aplicació, poder seleccionar un fitxer de vídeo en format H264 i quan s'hagi seleccionat s'ha de començar a descodificar els píxels, cada vegada que obtenim aquest s'ha de renderitzar a una superfície per tal de poder veure el vídeo. A més s'ha de poder pausar aquesta reproducció i una vegada acabat s'ha de poder tancar la aplicació o seleccionar un altre fitxer de vídeo.

Per contestar a la pregunta quin guany aconseguim necessitem que una mateixa aplicació utilitzi un descodificador basat en GPU i una altre basat en CPU i fer les comparacions necessàries amb els següents requisits:

- Utilitzar sempre el mateix dispositiu per fer les mesures.
- El overhead de totes les crides, que no depenguin de la descodificació, ha de ser pràcticament 0 o, com a mínim, igual a totes les mesures.
- Utilitzar el mateix tipus de vídeo entre proves, és a dir, la mateixa resolució i format.

Per desenvolupar aquestes mesures s'han de crear 2 aplicacions iguals o pràcticament iguals. Per fer la part de descodificació utilitzant la CPU, s'ha escollit els còdecs FFmpeg. Els motius d'aquesta elecció és la facilitat d'ús i que permet optimitzacions depenent del tipus de processador que volem utilitzar. Per fer la part de descodificador per GPU s'ha escollit utilitzar OpenMAX AL, tal i com s'indica en l'apartat d'OpenMAX AL, hi ha una implementació relativament fàcil d'utilitzar a Android. A part de treure mesures podem comprovar el funcionament d'OpenMAX a nivell d'AL i quins components d'Android interactuant.

En el següent esquema podem veure com dividirem les nostres aplicacions i les parts en comú que tenen cadascuna.

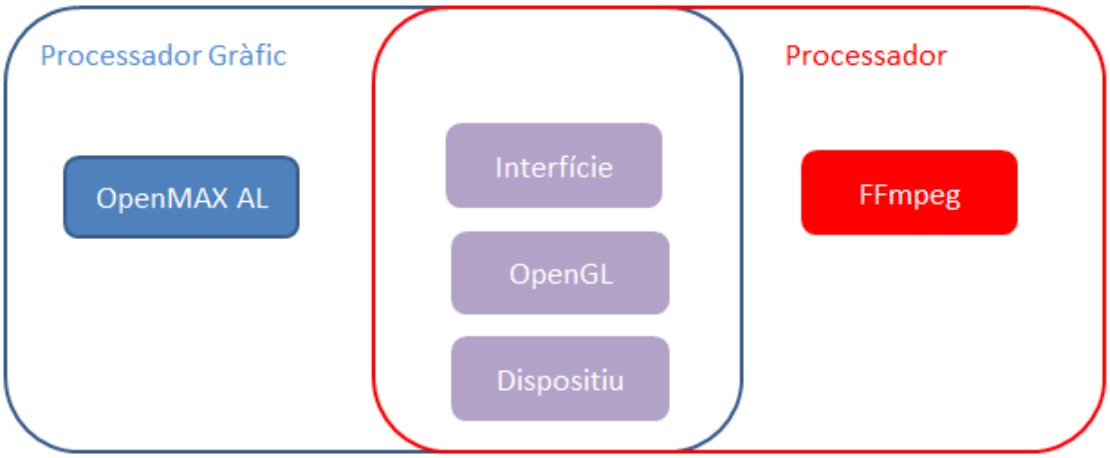

Figura 24: Divisió de les 2 aplicacions desenvolupades

Una vegada implementades les aplicacions s'ha de fer els experiments i treure conclusions. Els objectius d'aquestes conclusions són:

- Comprovar l'ús de la GPU en les crides d'OpenMAX.
- Comprovar l'ús de la CPU en les crides FFmpeg.
- Comprovar l'estalvi d'ús de CPU al utilitzar la GPU.
- Quantificar aquest estalvi.
- Comprovar el correcte funcionament de les aplicacions.

4.1 Aplicació utilitzant GPU

Com a primera aproximació i per entendre com funciona OpenMAX s'ha creat una aplicació fent servir OpenMAX AL amb l'ajut del NDK d'Android. Recordar que OpenMAX AL a Android és una versió molt limitada que no dóna suport pràcticament a cap protocol estàndard i no permet la configuració del descodificadors hardware dels dispositius tot i que l'utilitza.

Com s'ha dit a l'apartat anterior aquesta aplicació es basarà en un reproductor que obrirà un arxiu H264, el descodificarà i el reproduirà.

4.1.1 Resultats esperats

Una vegada implementada aquesta aplicació esperem que tingui un consum baix de la CPU, ja que no tenim cap altre aplicació per fer comparacions i treure conclusions, només esperarem un consum baix de la CPU sense poder dir quin percentatge de CPU utilitzarà aquesta aplicació. Aquest petit ús ens indicaria que estem utilitzant la GPU del dispositiu i estem alliberant la CPU per altres tasques.

No esperem una caiguda de frames per segon ni una pèrdua de qualitat de vídeo ja que el dispositiu hauria de ser capaç de suportar aquesta resolució.

La funcionalitat d'aquesta aplicació serà semblant a la que podem trobar a la llibreria estàndard MediaPlayer d'Android amb una millora de rendiment respecte aquesta. I ha de complir aquestes funcionalitats sense cap tipus d'error.

4.1.2 Implementació

S'ha de recordar que OpenMAX AL està desenvolupat completament en C, però es pot cridar indiferentment des de codi C or C++, és per això que en el nostre codi inclourem les següents crides a llibreries en format C++:

```
#include <OMXAL/OpenMAXAL.h>
#include <OMXAL/OpenMAXAL_Android.h>
```

Codi 1: Capçaleres de OpenMAX AL

Per desenvolupar aquesta aplicació s'ha de tenir en compte la distinció que OpenMAX AL té sobre els objectes i les interfícies i la seva seqüencia d'inicialització. Un objecte nomes és visible a traves de la seva interfície associada (quan es crea una objecte retorna un punter a aquest XAObjectItf). Aquesta distinció és fa degut a que la construcció de l'objecte no pot fallar (excepte per falta de memòria o per paràmetres incorrectes), en canvi al crear la interfície pot fallar per falta de recursos. Això també implica que es pugui associar més recursos si són necessaris. Aquestes interfícies es diuen interfícies dinàmiques que permeten no especificar les interfícies necessàries quan es crea l'objecte.

Després de la creació de cada objecte, mitjançant la crida GetInterface, l' objecte adquireix els recursos que són necessaris. Finalment l'objecte està preparat per ser accedit mitjançant les interfícies.

El fragment de codi referent a la inicialització és el següent:

```
NativeMedia_createEngine(JNIEnv* env, jclass clazz) {
   XAresult res;
   res = xaCreateEngine(&engineObject, 0, NULL, 0, NULL, NULL);
   assert(XA_RESULT_SUCCESS == res);
   res = (*engineObject)->Realize(engineObject, XA_BOOLEAN_FALSE);
   assert(XA_RESULT_SUCCESS == res);
   res = (*engineObject)->GetInterface(engineObject, XA_IID_ENGINE, &engineEngine);
   assert(XA_RESULT_SUCCESS == res);
   res = (*engineEngine)->CreateOutputMix(engineEngine, &outputMixObject, 0, NULL, NULL);
   assert(XA_RESULT_SUCCESS == res);
   res = (*outputMixObject)->Realize(outputMixObject, XA_BOOLEAN_FALSE);
   assert(XA_RESULT_SUCCESS == res);
res = (*engineEngine)->CreateMediaPlayer(engineEngine, &playerObj, &dataSrc, NULL,
&audioSnk, &imageVideoSink, NULL, NULL, NB_MAXAL_INTERFACES, iidArray required);
assert(XA_RESULT_SUCCESS == res);
res = (*playerObj)->Realize(playerObj, XA_BOOLEAN_FALSE);
assert(XA_RESULT_SUCCESS == res);
res = (*playerObj)->GetInterface(playerObj, XA_IID_PLAY, &playerPlayItf);
assert(XA_RESULT_SUCCESS == res);
```

```
res = (*playerObj)->GetInterface(playerObj, XA_IID_STREAMINFORMATION,
&playerStreamInfoItf);
assert(XA_RESULT_SUCCESS == res);
res = (*playerObj)->GetInterface(playerObj, XA_IID_VOLUME, &playerVolItf);
assert(XA_RESULT_SUCCESS == res);
res = (*playerObj)->GetInterface(playerObj, XA_IID_ANDROIDBUFFERQUEUESOURCE,
&playerBQItf);
assert(XA_RESULT_SUCCESS == res);
}
```

Codi 2: Inicialització d'OpenMAX AL

Una vegada creat, inicialitzat l'objecte i creat les seves interfícies (recursos), només ens queda registrar les crides de la manera següent:

```
  res = (*playerBQItf)->RegisterCallback(playerBQItf, AndroidBufferQueueCallback, NULL);
  assert(XA_RESULT_SUCCESS == res);
```

Codi 3: Registrar les crides d'OpenMAX AL

Ara ja està tot el treball inicial fet i ja podem canviar l'estat del reproductor a parat (PAUSED) o reproduint (PLAYING). En aquest moment és quan es connecten els recursos, ja que, com s'ha dit anteriorment, al fer la inicialització de la interfície estem assignant recursos, però no els estem connectant. Al següent fragment de codi podem veure com es canvia l'estat:

```
setPlaying (JNIEnv* env, jclass clazz, jboolean isPlaying) {
  XAresult res;
  if (NULL != playerPlayItf) {
    res = (*playerPlayItf)->SetPlayState(playerPlayItf, isPlaying ?XA_PLAYSTATE_PLAYING :
XA_PLAYSTATE_PAUSED);
  }
}
```

Codi 4: Canvi d'estat a d'OpenMAX AL

Una nota important que s'ha de tenir molt en compte, és que les crides són asíncrones respecte a l' aplicació, això vol dir que necessitem utilitzar semàfors o altres mecanisme de sincronització (mutex) per tal de controlar les variables compartides entre l'aplicació i les crides i evitar els possibles erros de lectura i escriptura. Un exemple d'això és la variable compartida del buffer, que necessitem aplicar aquest mecanisme d'exclusió mútua per a què no es modifiqui la variable en dos parts diferents del codi, en el següent fragment de codi es mostra la sincronització d'aquesta variable buffer:

```
if (NULL != playerBQItf && NULL != file) {
    int ok;
    ok = pthread_mutex_lock(&mutex);
    assert(0 == ok);
    discontinuity = JNI_TRUE;
    while (discontinuity && !reachedEof) {
       ok = pthread_cond_wait(&cond, &mutex);
       assert(0 == ok);
    }
    ok = pthread_mutex_unlock(&mutex);
    assert(0 == ok);
 }
```

Codi 5: Semàfor implementat a d'OpenMAX AL

Per últim, ens queda destruir els objectes creats abans de sortir de la nostre aplicació. La destrucció dels objectes s'ha de fer en ordre invers a la seva creació per tal d'evitar possibles errors de dependència entre els objectes.

OpenMAX AL no disposa de col·lector de brossa com Java, ja que esta escrit en llenguatge C, per tant aquest treball s'ha de fer manualment. Després de destruir un objecte totes les extensions derivades d'aquest passen a estat indefinit. Per tal de que quan tornem a executar l'aplicació no es produeixin errors accidentals és molt recomanable associar valors nuls a les interfícies després de destruir l'objecte.

```
if (playerObj != NULL) {
    (*playerObj)->Destroy(playerObj);
    playerObj = NULL;
    playerPlayItf = NULL;
    playerBQItf = NULL;
    playerStreamInfoItf = NULL;
    playerVolItf = NULL;
}
if (outputMixObject != NULL) {
    (*outputMixObject)->Destroy(outputMixObject);
    outputMixObject = NULL;
}
if (engineObject != NULL) {
    (*engineObject)->Destroy(engineObject);
    engineObject = NULL;
    engineEngine = NULL;
}
```

Codi 6: Destrucció dels components a d'OpenMAX AL

Una vegada desenvolupat la part important d'OpenMAX AL s'ha desenvolupat la interfície gràfica, tot i que desenvolupar aquesta part gràfica i les funcionalitats principals d'una aplicació Android requereix temps (crear les activitats, associar els permisos de seguretat necessaris...), en aquest document no entrarem amb detall en aquesta explicació ja que no és el principal objectiu d'aquest projecte.

4.1.3 Resultats finals

Una vegada acabada tota la implementació i comprovada que funciona correctament, podem dir que hem aconseguit descodificar un vídeo H264 i reproduint-lo correctament. A part la aplicació permet pausar, avançar i reproduir el vídeo sense cap error aparent.

Una vegada comprovat que l'aplicació no te cap errata s'ha provat amb vídeos de diferents qualitats extremes. En aquestes proves s'ha pogut veure que el consum de la CPU és del 14% - 16% del dispositiu, si el vídeo és de qualitat molt baixa es pot apreciar un consum de la CPU del 10%.

En les següents taules es mostra les proves realitzades i els resultats d'aquestes proves.

Nom	**Ample**	**Alçada**	**Format**	**FPS**	**% CPU**
Vídeo 1	160	122	H264	21,82	9,26%
Vídeo 2	800	600	H264	20,90	14,04%
Vídeo 3	1920	1088	H264	7,30	14,72%

Taula 8: Dades extretes de l'aplicació amb OpenMAX AL

	Mostra 1		**Mostra 2**		**Mostra 3**		**Mostra 4**		**Mostra 5**	
Nom	**FPS**	**% CPU**	**FPS**	**% CPU**	**FPS**	**% CPU**	**FPS**	**% CPU**	**FPS**	**% CPU**
Vídeo 1	21,30	9,30%	22,40	9,10%	22,30	9,20%	21,80	9,30%	21,30	9,40%
Vídeo 2	23,40	14,20%	21,40	14,40%	21,30	14,20%	20,40	13,30%	18,00	14,10%
Vídeo 3	7,30	14,40%	8,40	13,90%	7,40	15,10%	8,00	14,90%	5,40	15,30%

Taula 9: Dades extretes de l'aplicació amb OpenMAX AL

Més tard aquest resultats els compararem amb una aplicació que només utilitzi descodificació via CPU i podrem obtenir més conclusions.

A aquest apartat es remarcà un resultat que al principi va ser bastant sorprenent. Com s'ha dit anteriorment les funcionalitats que es volen són les mateixes que la classe MediaPlayer d'Android. Per tal de comprovar que s'ha aconseguit aquestes funcionalitats es va fer un petit experiment amb la classe MediaPlayer.

Aquesta classe és relativament fàcil d'utilitzar i no entrarem en detall en com es va crear. Al següent fragment de codi es mostra les operacions que calen fer:

```
MediaPlayer mediaPlayer = new MediaPlayer();

mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);

mediaPlayer.setDataSource(getApplicationContext(), URL);

mediaPlayer.prepare();

mediaPlayer.start();
```

Codi 7: Codi utilitzant la classe MediaPlayer d'Android

La sorpresa de l'experiment va ser descobrir que obtenim els mateixos resultats que utilitzant OpenMAX AL, tot i que podríem pensar que tractant-se una classe Java no s' utilitza la GPU del dispositiu. Per tal d'aconseguir esbrinar que està passant s'ha necessitat utilitzar l'eina profiling per seguir una traça d'execució. Els passos per aconseguir fer profiling d'una aplicació Android són els següents:

Modificar el fitxer Android.mk per importar el mòdul del profiler del NDK d'Android, enllaçar aquesta amb la biblioteca corresponent i compilar el codi amb la informació corresponent.

```
# compile with profiling

LOCAL_CFLAGS := -pg

LOCAL_STATIC_LIBRARIES := android-ndk-profiler

# at the end of Android.mk

$(call import-module,android-ndk-profiler)
```

Codi 8: Modificació del Android.mk per fer profiling

S'ha d'afegir els permisos perquè una aplicació pugui escriure a la memòria del dispositiu, ja que el fitxer gmon.out el guardarem a la memòria. Per canviar aquest permisos s'ha d'escriure al manifest d'Android la següent línia:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Codi 9: Modificació dels permisos d'Android per fer profiling

Una vegada modificat això, ja es pot descarregar el fitxer gmon.out de la memòria i interpretar les dades.

Aquestes dades del profiling han demostrat el perquè la classe MediaPlayer ha de tenir el mateix rendiment que una aplicació utilitzant OpenMAX. Això es degut a que a partir de la versió 4.0 d'Android, Google ha incorporat que la classe MediaPlayer utilitza OpenMAX i aquesta els còdecs de descodificació hardware. Al següent esquema es mostra com és l'arquitectura a partir de la versió 4.0 d'Android:

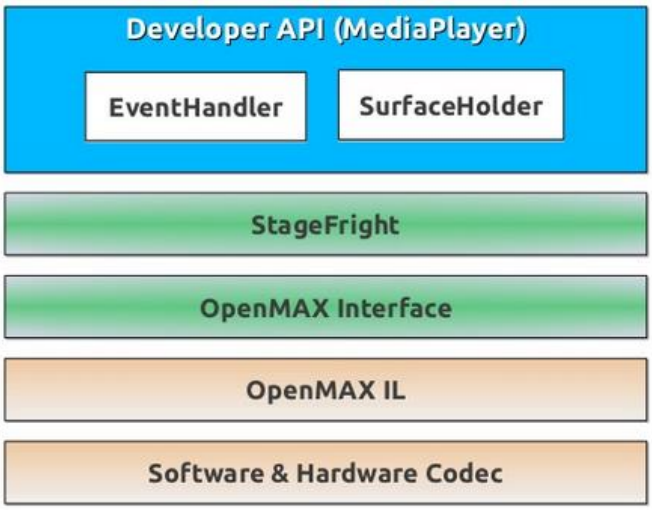

Figura 25: Arquitectura MediaPlayer d'Android

StageFright és un reproductor creat per Google per facilitar l'ús de la descodificació per software i hardware. En la següent figura es mostra amb detall com és la implementació d'aquest reproductor pel que fa a classes Java:

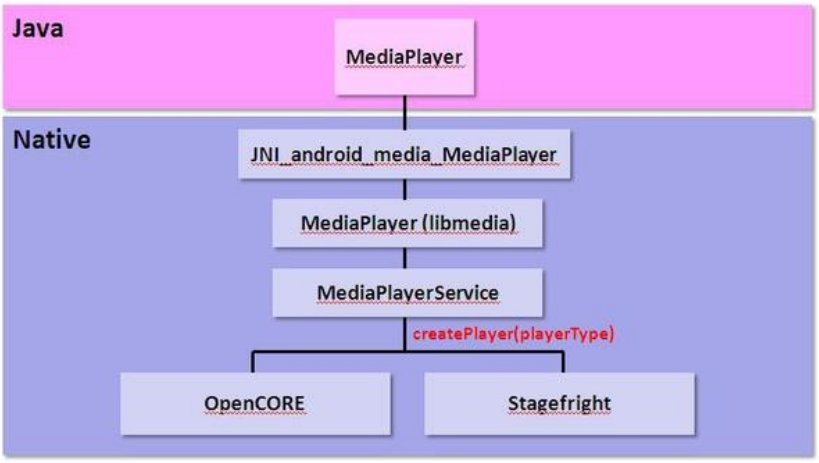

Figura 26: Arquitectura StageFright *d'Android*

Ara be la pregunta és: per què no utilitzar aquesta classe que és fàcil d'utilitzar, ja està integrada al framework d'Android i utilitza la GPU? La resposta és senzilla, per la seva no compatibilitat. La versió 4.1 d'Android, que és la més actual, només suporta 3 tipus de format de vídeo (H264, H263 i MPEG-4) i no suporta tot tipus de transports d'aquest formats com per exemple MPEG-TS. També depenen del tipus de dispositiu aquest format pot no ser compatible. Per suposat no es possible utilitzar-lo en versions anteriors a la 4.0. I per últim, les possibilitats de configuració són mínimes.

És per aquesta raó de falta de compatibilitat i falta de possibilitat de configuració, que aquesta classe no és útil per als objectius d'aquest projecte.

4.2 Aplicació utilitzant CPU

Com hem vist abans no podem utilitzar les classes de Java per fer una aplicació que descodifiqui l'H264 utilitzant només la CPU, almenys a versions posteriors a la 4.0. Per tal necessitem un altre mètode que ens permeti estar segurs que nomes utilitzarem la CPU. Per poder desenvolupar aquesta aplicació s'ha utilitzat FFmpeg i OpenGL per renderitzar el vídeo. S'ha utilitzat aquesta opció perquè és relativament senzilla d'implementar i compatible amb diferents versions Android.

FFmpeg és una col·lecció de software lliure que permet gravar, codificar, descodificar i reproduir vídeo i àudio. FFmpeg inclou una llibreria de còdecs anomenada libavcodec (que és la que utilitzem per aquest projecte). FFmpeg està dissenyat per a Linux sota llicencia GNU, però és compatible per a tots els sistemes operatius.

Figura 27: Logotip de FFmpeg

També s'ha escollit per fer la part de renderitzar el vídeo OpenGL (tot i que no forma part dels objectius d'aquest projecte), d'aquesta manera una vegada que s'ha aconseguit que el OpenGL funcioni, es pot reutilitzar per a totes les futures aplicacions de proves d'aquest projecte.

A més, abans de començar la implementació d'aquesta aplicació, s'ha posat com a objectiu poder desactivar la renderització del vídeo per tal de poder prendre mesures sense el overhead de les crides d'OpenGL.

4.2.1 Resultats esperats

Com totes les proves, primer de tot esperem que podem reproduir el vídeo sense cap desperfecte i que l' aplicació no doni cap tipus d'error.

A aquesta prova esperem que tot el càlcul de descodificació es faci des de la part de CPU, per tant al utilitzar només la CPU podem dir que el rendiment del dispositiu ha de disminuir, és a dir, ha de augmentar l'ús de la CPU. A més a més, s'espera que el dispositiu en certs vídeos

d'alta qualitat no el pugui arribar a descodificar completament i es transformi en una caiguda de frames per segons o fins i tot es bloquegi.

4.2.2 Implementació

Abans de començar en detall amb la implementació, s'ha de decidir com serà la nostra aplicació per tal que s'adapti completament a FFmpeg. Al següent codi d'alt nivell es mostra l'esquema general de la nostre aplicació:

```
Registrar_codecs()

Obrir_fitxer(fitxer.h264)

WHILE paquet <> final THEN

        Frames = Llegir_paquet (paquet)

        Descodificar_frame(Frames)

Tancar_fitxer()

Tancar_codecs()
```

Codi 10: Codi en alt nivell del funcionament general de FFmpeg

Una vegada es té clar l'estructura de la nostra aplicació només ens cal començar amb la implementació. FFmpeg és una classe singletó, per tant hem d'accedir aquesta classe tal i com es mostra al fragment de codi:

```
        public static FFmpeg getInstance() {
                if( singleton == null )singleton = new FFmpeg();
                return singleton;
        }
```

Codi 11: Utilització del patró singletó a FFmpeg

Durant la fase d' implementació utilitzarem les següents crides de FFmpeg que explicarem amb detall. Aquestes crides són estàndards de FFmpeg i en cas de que donin error retornaran el resultat -1.

private native void *void avRegisterAll();*

Funció que registra tots els possibles còdecs a utilitzar, això fa que quan s'obri un fitxer que necessiti un còdec concret automàticament es carregui aquest còdec a la memòria. Sense utilitzar aquesta funció s'hauria d'anar registrant còdec a còdec, això inclou els còdecs, els parsers, els filtres, etc.

private native boolean avOpenInputFile(String fitxer);

Funció que s'encarrega d'obrir un fitxer concret. El paràmetre d'aquesta funció és la ruta al fitxer de vídeo que volem descodificar. Si es vol obrir un altre fitxer s'ha de tancar primer aquest.

private native void avCloseInputFile();

Aquesta funció s'encarrega de tancar el fitxer d'entrada, aquesta funció s'ha de cridar al final del codi o quan es vol descodificar un altre fitxer. Cal remarcar que no tanca el còdec utilitzat.

private native boolean avFindStreamInfo();

Aquesta funció ens mostra en quin format esta el vídeo i quina és la seva resolució. Tal i com mostra en el següent exemple:

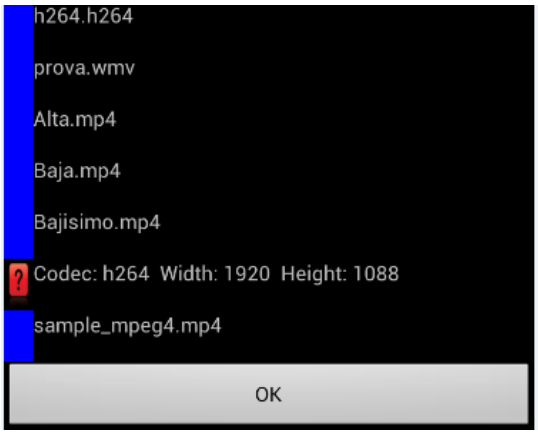

Figura 28: Exemple de la informació de l'aplicació FFmpeg

private native boolean avcodecFindDecoder();

Aquesta funció es crida sempre després d'obrir el fitxer i el resultat és que automàticament busca el còdec necessari per tractar el fitxer que s'ha obert.

private native void avcodecAllocFrame();

Durant el procés de descodificació necessitem guardar frame, degut que necessitem convertir els nostres frames a un format específic en el nostre cas RGB. Amb aquesta funció assigna a memòria un frame.

private native void avFree();

Aquesta funció desassigna de la memòria el frame reservat per fer les conversions. S'ha de cridar una vegada fet tot el treball. Recordem que en aquesta part del codi no tenim col·lector de brossa.

private native int avcodec_decode_video(AVCodecContext *codec, AVFrame *frames_sortida, int *resultat, uint8_t *buffer_entrada, int tamany_buffer)

Funció que descodifica un frame del buffer de entrada. Per descodificar aquest frame utilitza el còdec que ha estat obert amb la funció avcodecFindDecoder(). Si la funció no ha pogut descodificar aquest frame el frame de sortida serà 0.

private native void avcodecClose();

Aquesta funció tanca tots els còdecs oberts. S'ha de cridar al final del nostre codi.

Ara nomes ens queda compilar les crides FFmpeg per adaptar-les al nostre dispositiu Android. Per simplificar aquesta part es va utilitzar un script estàndard que esta inclòs en les mateixes crides de FFmpeg i només es va afegir que compiles el còdec per descodificar H264 i les optimitzacions oportunes.

Primer de tot es va compilar sense optimitzacions, però sense optimitzacions el rendiment va ser molt baix i només va ser capaç de reproduir el vídeo en el triple de temps necessari. Per això es va decidir compilar-lo amb optimitzacions, les optimitzacions principals que s'han utilitzat són: la compilació especifica per a arquitectura ARM v7 i la utilització de les instruccions NEON (vectorització).

NEON és un SIMD (una instrucció moltes dades) que porta el processador. Amb NEON els registres es poden considerar vectors d'elements del mateix tipus de dades. Afavorint així el rendiment del dispositiu.

El següent fragment de codi és el script que hem construït per compilar FFmpeg, com es pot apreciar s'ha activat totes les possibles millores de FFmpeg.

```
#!/bin/bash
NDK=~/Desktop/android/android-ndk
PLATFORM=$NDK/platforms/
PREBUILT=$NDK/toolchains/arm-linux-androideabi-4.4.3/prebuilt/linux-x86
function build_one
{
./configure --target-os=linux \
   --prefix=$PREFIX \
   --enable-cross-compile \
   --extra-libs="-lgcc" \
   --arch=arm \
   --cc=$PREBUILT/bin/arm-linux-androideabi-gcc \
   --cross-prefix=$PREBUILT/bin/arm-linux-androideabi- \
   --nm=$PREBUILT/bin/arm-linux-androideabi-nm \
   --sysroot=$PLATFORM \
   --extra-cflags=" -O3 -fpic -DANDROID -DHAVE_SYS_UIO_H=1 -Dipv6mr_interface=ipv6mr_ifindex -fasm -Wno-psabi -fno-short-enums -fno-strict-aliasing -finline-limit=300 $OPTIMIZE_CFLAGS " \
   --disable-shared \
   --enable-static \
   --extra-ldflags="-Wl,-rpath-link=$PLATFORM/usr/lib -L$PLATFORM/usr/lib -nostdlib -lc -lm -ldl -llog" \
   --disable-everything \
   --enable-demuxer=mov \
   --enable-demuxer=h264 \
   --disable-ffplay \
```

```
  --enable-protocol=file \
  --enable-avformat \
  --enable-avcodec \
  --enable-decoder=mpeg4 \
  --enable-decoder=h264 \
  --enable-parser=h264 \
  --disable-network \
  --enable-zlib \
  --disable-avfilter \
  --disable-avdevice \
  $ADDITIONAL_CONFIGURE_FLAG
make clean
make  -j4 install
$PREBUILT/bin/arm-linux-androideabi-ar d libavcodec/libavcodec.a inverse.o
$PREBUILT/bin/arm-linux-androideabi-ld -rpath-link=$PLATFORM/usr/lib -L$PLATFORM/usr/lib -soname libffmpeg.so -shared -nostdlib  -z,noexecstack -Bsymbolic --whole-archive --no-undefined -o $PREFIX/libffmpeg.so libavcodec/libavcodec.a libavformat/libavformat.a libavutil/libavutil.a libswscale/libswscale.a -lc -lm -lz -ldl -llog  --warn-once  --dynamic-linker=/system/bin/linker $PREBUILT/lib/gcc/arm-linux-androideabi/4.4.3/libgcc.a
}
CPU=armv7
OPTIMIZE_CFLAGS="-mfloat-abi=softfp -mfpu=neon -marm -march=$CPU -mtune=cortex-a8"
PREFIX=./android/$CPU
ADDITIONAL_CONFIGURE_FLAG=--enable-neon
build_one
```

Codi 12: Makefile de l'aplicació de FFmpeg

4.2.3 Resultats finals

El reproductor disposa de totes les funcionalitats esperades, el vídeo es reprodueix de principi al final sense cap error greu. En aquest cas s'han complert els resultats esperats ja que el consum de la CPU ha sigut força alt o, com a mínim, més alt que la prova anterior utilitzant GPU.

Per les característiques del dispositiu de gamma alta i utilitzant FFmpeg amb les optimitzacions de compilació per dispositius de ARM v7, s'ha aconseguit reproduir vídeos d'alta qualitat (600x400) a uns 30 frames per segon. Però el consum de la CPU és elevat, al menys és més elevat respecte a l'aplicació utilitzant OpenMAX, degut la CPU és de doble nucli i degut a què el codi que hem desenvolupat no és multithread dir que consumeix un 40-50% de la CPU és com dir pràcticament que hi ha un nucli de la CPU està pràcticament utilitzat per aquesta tasca de descodificació.

Cal dir que també s'ha fet aquestes proves sense renderitzar la imatge, i ha donat resultats molt semblants. A continuació es mostra els resultats obtinguts amb les 3 diferents qualitats de vídeo i una comparació entre elles.

El primer vídeo que s'ha escollit és de qualitat molt baixa (160x112), com es pot veure a la següent imatge:

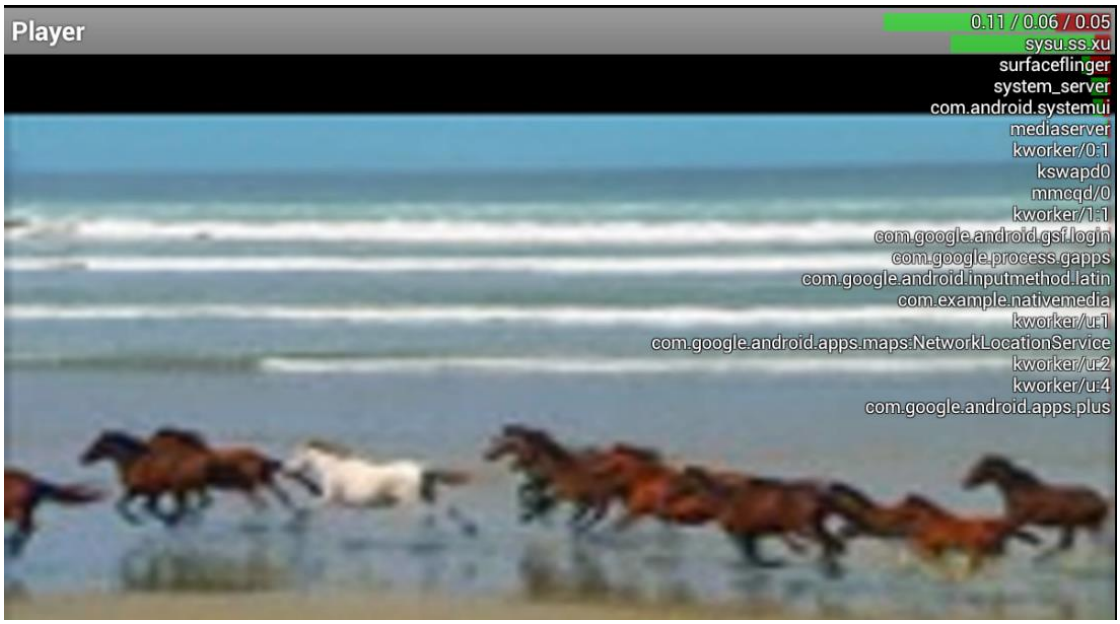

Figura 29: Imatge del vídeo 160x122 a la aplicació FFmpeg

El segon vídeo que s'ha escollit és de qualitat alta (800x600) i obtenim el següent resultat:

Figura 30: Imatge del vídeo 800x600 a la aplicació FFmpeg

Per últim s'ha escollit un vídeo de qualitat molt alta (1920x1088), aquest vídeo té més resolució que el propi dispositiu per tant a alhora de fer el render de la imatge perd qualitat, en la següent imatge mostra el resultat:

Figura 31: Imatge del vídeo 1920x1088 a la aplicació FFmpeg

Per treure les següents mesures s'ha utilitzat la comanda "top" de Linux, com es pot veure en les imatges anteriors, al requadre de dalt a la esquerre també indica el consum de CPU i els processos actius, però hi ha un retard en l' actualització d'aquest resultat per tant els valors poden diferir lleugerament. Per aquests motius els valors mostrats a continuació estan obtinguts amb la comanda top. A continuació es mostra un recull dels valors obtinguts.

Nom	**Ample**	**Alçada**	**Format**	**FPS**	**% CPU**
Vídeo 1	160	122	H264	28,88	11,78%
Vídeo 2	800	600	H264	27,30	17,40%
Vídeo 3	1920	1088	H264	8,46	53,06%

Taula 10: Dades extretes de l'aplicació amb FFmpeg

	Mostra 1		**Mostra 2**		**Mostra 3**		**Mostra 4**		**Mostra 5**	
Nom	**FPS**	**% CPU**	**FPS**	**% CPU**	**FPS**	**% CPU**	**FPS**	**% CPU**	**FPS**	**% CPU**
Vídeo 1	28,00	11,80%	30,10	12,00%	29,30	11,50%	28,70	11,60%	28,30	12,00%
Vídeo 2	27,10	17,00%	27,30	16,00%	28,10	17,00%	26,70	18,00%	27,30	19,00%
Vídeo 3	9,00	54,30%	8,30	51,00%	7,50	53,00%	9,10	54,00%	8,40	53,00%

Taula 11: Dades extretes de l'aplicació amb FFmpeg

Al provar l'aplicació amb el dispositiu alternatiu (Galaxy Mini amb 2.3) de gamma baixa, no pot descodificar el vídeo a aquesta resolució i es bloqueja. Per tant no he pogut treure dades amb aquest dispositiu.

4.2.4 Experiments: Comparacions i conclusions

En aquesta secció es comparà les 2 aplicacions creades amb els resultats obtinguts i es respondran a les preguntes que s'ha proposat a la secció inicial.

Recordem els tres vídeos de diferents qualitats que s'ha utilitzat per fer els experiments oportuns.

Nom	Ample	Alçada	Format
Vídeo 1	160	122	H264
Vídeo 2	800	600	H264
Vídeo 3	1920	1088	H264

Taula 12: Recull dels vídeos utilitzats de proves

A la següent taula i als següents gràfics es mostra la comparació entre les dues aplicacions. Compararem les dues aplicacions en la mesura de l'ús de la CPU i frames per segon que aconsegueixi el vídeo.

	CPU		GPU	
CPU	**FPS**	**% CPU**	**FPS**	**% CPU**
Vídeo 1	28,88	11,78%	21,82	9,26%
Vídeo 2	27,30	17,40%	20,90	14,04%
Vídeo 3	8,46	53,06%	7,30	14,72%

Taula 13: Comparació de les 2 aplicacions desenvolupades

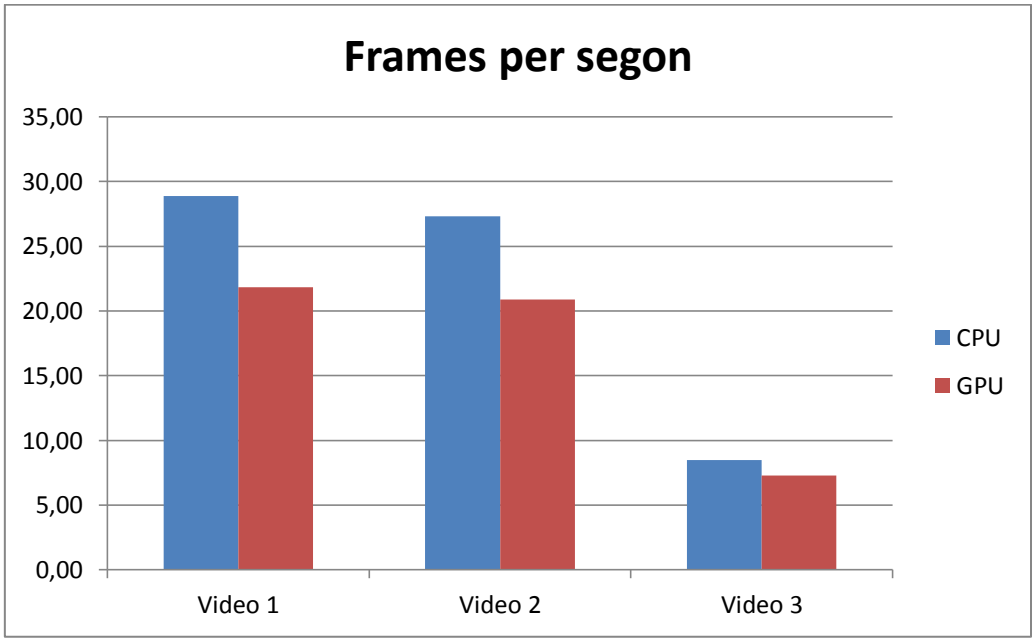

Figura 32: Gràfic comparatiu de frames per segon entre les 2 aplicacions desenvolupades

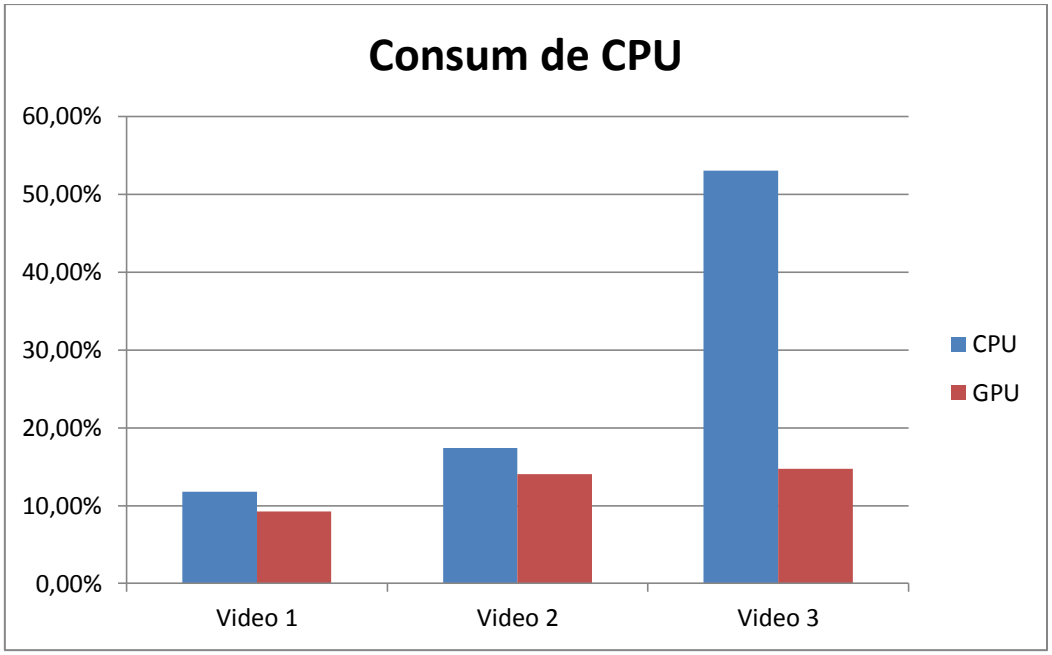

Figura 33: Gràfic comparatiu de consum de la CPU entre les 2 aplicacions desenvolupades

Podem observar un consum de CPU, inferior si utilitzem el coprocessador gràfic per fer la descodificació del vídeo. Aquesta tendència s'accentua quan el vídeo és de més qualitat. A partir dels 800x600 el consum de la CPU creix exponencialment. A qualitats de vídeo normals (800x600) estem parlant d'un guany del 18% y la pèrdua de fps no és significativa. Per notar diferències importants el vídeo ha de ser de qualitat molt alta.

Recordem que les crides FFmpeg utilitzades per crear l'aplicació d'ús de la CPU, estan optimitzades pels processadors arm v7 amb el flag NEON, és a dir, que estan específicament compilades i optimitzades pel nostre dispositiu concret.

Per tant es pot assumir que crear una aplicació basada en OpenMAX que utilitzi el descodificador hardware del dispositiu té els avantatges que hem suposat respecte al no utilitzar-lo. Per tant el següent i últim pas d'aquest projecte és la construcció d'aquesta API de descodificador de vídeo mitjançant el hardware.

5. Nova API de descodificació de vídeo

En aquest apartat s'explicarà com s'ha arribat a la creació de la nostre API, recordem que tot el treball previ ha sigut per provar que aquesta API és necessària i molt útil per a futurs desenvolupadors d'aplicacions multimèdia d'Android. També el treball previ fet ens facilitarà les proves d'aquesta API, ja que es disposa d'un reproductor lleuger i de les crides d'OpenGL necessàries ja implementades. Aquest apartat està dividit en 3 fases: especificació i disseny, implementació i proves.

En la especificació de requisits es farà una descripció completa del comportament de la API que es desenvoluparà. El disseny és la fase mes important del projecte ja que un bon disseny de l'API millora el temps invertit a les següents fases. En aquest apartat també es farà una explicació profunda d'OpenMAX IL.

En la implementació desenvoluparem la nostre API, aquesta implementació s'ha fet com s'explica en aquest apartat.

A les proves utilitzarem el reproductor de l'estudi previ i s'ha de comprovar que s'aconsegueixen els mateixos resultats que a les proves anteriors.

5.1 Especificació i disseny

L'API s'ha decidit que fos una llibreria d'ús dinàmic (format .so) que es pugui accedir des de el jni d'Android. Donat que OpenMAX està desenvolupat íntegrament en C, la nostre API també estarà desenvolupada íntegrament en C. A la següent figura es pot veure com serà l'estructura de comunicació. Podem veure que les aplicacions desenvolupades es comunicaran amb la nostre API de descodificació de vídeo i aquesta es comunicarà amb el nucli d'OpenMAX IL, on es crearan els components que accediran als còdecs necessaris.

Abans de començar amb la especificació, s'ha investigat per buscar API de descodificació de vídeo al mercat actual per utilitzar-la com a referència. A data de Juliol 2012 hi ha 8 APIs importants de descodificació per diverses plataformes. Entre elles destaquen DXVA (DirectX Video Acceleration). Aquesta API ha estat dissenyada per Microsoft i està especialment enfocada a dispositius Windows i Xbox. Un altra API que destaca és VAAPI (VideoAcceleration

API) inicialment dissenyada i desenvolupada per Intel en 2007, està enfocada en sistemes Windows i Unix.

Però després de estudiar totes elles i veure que són semblants pel que fa a continguts, la més simple, amb més documentació i que té les crides més clares és VDADecoder. Per tant la API que s'ha utilitzat com a referència és VDADecoder, creada per Apple per als sistemes operatius Mac a partir de la versió OS X 10.6.3 i superiors.

Al utilitzar aquesta API com ha referència crearem les nostres estructures i funcions amb capçaleres semblants per no crear confusions. També cal dir que s'han eliminat funcions i estructures no necessàries per la creació de la nostre API, com poden ser les funcions de codificació de vídeo o funcions més avançades per controlar el flux de descodificació, ja que no la necessitem per aquest projecte.

L' estructura d'un frame, ja sigui de sortida o de entrada, a VDADecoder és la següent:

```
typedef struct myDisplayFrame {
        int64_t frameDisplayTime;
        CVPixelBufferRef  frame;
        struct myDisplayFrame *nextFrame;
} myDisplayFrame, *myDisplayFramePtr;
```

Codi 13: Estructura d'un frame a VDADecoder

```
//Creació del descodificador
OSStatus VDADecoderCreate(
        CFDictionaryRef             decoderConfiguration,
        CFDictionaryRef             destinationImageBufferAttributes,
        VDADecoderOutputCallback    *outputCallback,
        void                        *decoderOutputCallbackRefcon,
        VDADecoder                  *decoderOut)

//descodifica els frames
OSStatus VDADecoderDecode(
```

```
            VDADecoder          decoder,
            uint32_t            decodeFlags,
            CFTypeRef           compressedBuffer,
            CFDictionaryRef     frameInfo)

//Cancel·la tots els frames que no han estat completats
OSStatus VDADecoderFlush(VDADecoder decoder, uint32_t flushFlags)

//Destrueix el descodificador
OSStatus VDADecoderDestroy(VDADecoder decoder)
```

Codi 14: Especificacions de l'API VDADecoder

Resumint aquest és el recull de funcions que tindrà la nostre API i la seva equivalència a altres APIs del mercat (en aquest exemple a la dissenyada per Apple):

Aquesta API	**VDADecoder**	**Funcionalitat**
h264_create	VDADecoderCreate	Crea el component que s'encarregarà de la descodificació.
h264_decode	VDADecoderDecode	Descodifica frames d'un arxiu H264 determinat.
h264_cancel	VDADecoderFlush	Cancel·la els frames que no han estat encara descodificats.
h264_destroy	VDADecoderDestroy	Destrueix el component que s'encarregarà de la descodificació.

Taula 14: Especificació de l'API de descodificació

Més endavant en la implementació es veurà quins paràmetres tenen aquestes funcions però un paràmetre que hi serà en totes les funcions és el paràmetre corresponent al context del component.

Totes aquestes funcions retornaran 0 en cas d'èxit i en cas d'error es retornarà un codi d'error corresponent amb els codis d'errors especificats a OpenMAX IL. A continuació es pot trobar la capçalera amb les definició de tipus d'aquest errors:

```
typedef enum OMX_ERRORTYPE {

OMX_ErrorNone = 0,

OMX_ErrorInsufficientResources = 0x80001000,

OMX_ErrorUndefined = 0x80001001,

OMX_ErrorInvalidComponentName = 0x80001002,

OMX_ErrorComponentNotFound = 0x80001003,

OMX_ErrorInvalidComponent = 0x80001004,

OMX_ErrorBadParameter = 0x80001005,

OMX_ErrorNotImplemented = 0x80001006,

OMX_ErrorUnderflow = 0x80001007,

OMX_ErrorOverflow = 0x80001008,

OMX_ErrorHardware = 0x80001009,

OMX_ErrorInvalidState = 0x8000100A,

OMX_ErrorStreamCorrupt = 0x8000100B,

OMX_ErrorPortsNotCompatible = 0x8000100C,

OMX_ErrorResourcesLost = 0x8000100D,

OMX_ErrorNoMore = 0x8000100E,

OMX_ErrorVersionMismatch = 0x8000100F,

OMX_ErrorNotReady = 0x80001010,
```

```
OMX_ErrorTimeout = 0x80001011,

OMX_ErrorSameState = 0x80001012,

OMX_ErrorResourcesPreempted = 0x80001013,

OMX_ErrorPortUnresponsiveDuringAllocation = 0x80001014,

OMX_ErrorPortUnresponsiveDuringDeallocation = 0x80001015,

OMX_ErrorPortUnresponsiveDuringStop = 0x80001016,

OMX_ErrorIncorrectStateTransition = 0x80001017,

OMX_ErrorIncorrectStateOperation = 0x80001018,

OMX_ErrorUnsupportedSetting = 0x80001019,

OMX_ErrorUnsupportedIndex = 0x8000101A,

OMX_ErrorBadPortIndex = 0x8000101B,

OMX_ErrorPortUnpopulated = 0x8000101C,

OMX_ErrorComponentSuspended = 0x8000101D,

OMX_ErrorDynamicResourcesUnavailable = 0x8000101E,

OMX_ErrorMbErrorsInFrame = 0x8000101F,

OMX_ErrorFormatNotDetected = 0x80001020,

OMX_ErrorContentPipeOpenFailed = 0x80001021,

OMX_ErrorContentPipeCreationFailed = 0x80001022,

OMX_ErrorSeperateTablesUsed = 0x80001023,

OMX_ErrorTunnelingUnsupported = 0x80001024,

OMX_ErrorMax = 0x7FFFFFFF,

} OMX_ERRORTYPE;
```

Codi 15: Definicions del tipus d'error

En relació a l'arquitectura utilitzarem l'arquitectura en capes. Un desenvolupador només ha de conèixer l'aplicació que vol fer i la especificació de l'API de descodificador de vídeo, i a la implementació de l'API, feta en aquest projecte, ens encarregarem de crear totes les estructures necessàries d'OpenMAX i configurar tots els ports necessaris per tal de poder descodificar el vídeo. Com a norma general de l'arquitectura en capes que respectarem en aquest projecte, una aplicació mai ha de cridar directament capes inferiors que no sigui la capa immediatament inferior. Al nostre cas, les aplicacions mai cridaran directament a OpenMAX sense passar per la nostra API.

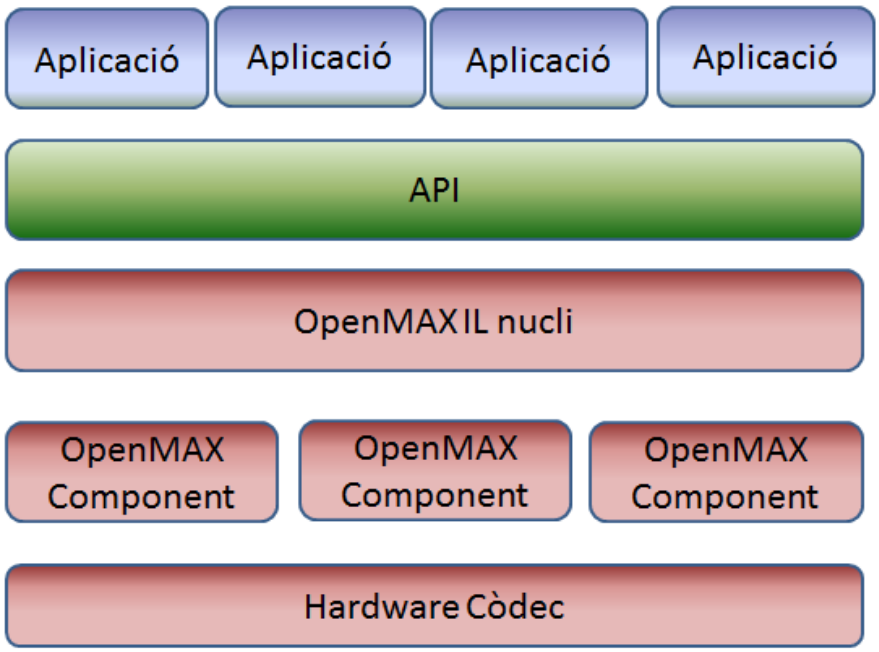

Figura 33: Arquitectura de la integració de l'API amb OpenMAX

5.2 Implementació

Abans de entrar amb detall a la implementació de la API, s'ha de crear el fitxer Android.mk que és el que compilarà la nostre llibreria.

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_MODULE := APIdesH264

LOCAL_SRC_FILES := APIdesH264.c

LOCAL_LDLIBS := -ldl  -llog

LOCAL_MODULE_TAGS := optional

LOCAL_CFLAGS := -DANDROID_NDK

include $(BUILD_SHARED_LIBRARY)

include $(CLEAR_VARS)

LOCAL_MODULE := APIdesH264

LOCAL_SRC_FILES := APIdesH264.c

LOCAL_LDLIBS := -ldl -llog

LOCAL_MODULE_TAGS := optional

LOCAL_CFLAGS := -DANDROID_NDK

include $(BUILD_EXECUTABLE)

include $(CLEAR_VARS)

LOCAL_MODULE := APIdesH264

LOCAL_SRC_FILES := APIdesH264_use.c

LOCAL_LDLIBS := -ldl -llog
```

```
LOCAL_MODULE_TAGS := optional

LOCAL_CFLAGS := -DANDROID_NDK

include $(BUILD_EXECUTABLE)
```

Codi 16: Contingut del fitxer Android.mk

Capçaleres

En aquest apartat es donarà una breu descripció de les capçaleres utilitzades.

```
#include <string.h>

#include <stdint.h>

#include <stdlib.h>

#include <stdio.h>

#include <jni.h>

#include <dlfcn.h>

#include <android/log.h>

#include <OMX_Core.h>

#include <OMX_Component.h>
```

Codi 17: Capçaleres necessàries de la nostre API

La capçalera “string.h” de la biblioteca estàndard del llenguatge de programació C que conté la definició de macros, constant, funcions per treballar amb cadena de caràcters. Les altres llibreries estàndards del llenguatge C són “stdint.h”, “stdlib.h” i “stdio.h”.

La capçalera “jni.h” es necessària per la programació de codi nadiu.

La capçalera “dlfcn.h” conté la macro necessària dlopen que permet carregar còdecs.

La capçalera “log.h” ens permet utilitzar la funció “android_log_print” que s’ha utilitzar en tot el codi per comprovar que OpenMAX no retorna cap error i que aconsegueix els resultats esperats. S’ha de tenir en compte que se li ha d’indicar al fitxer Android.mk que lligui amb la llibreria del log Android. (LOCAL_LDLIBS := -llog). Per accedir al log d’Android només ens cal utilitzar la funció “adb logcat” que ens imprimeix el log per pantalla.

La capçalera OMX_core.h conté totes les funcions necessàries per poder accedir a les funcions del nucli necessàries d’OpenMAX.

La capçalera OMX_Component.h conté totes les funcions necessàries per poder accedir a les funcions dels components necessàries d’OpenMAX.

Per poder utilitzar una estructura estàndard a tota la aplicació s’ha utilitzat un tipus d’estructura característica d’OpenMAX. Aquesta estructura permet guardar el context d’OpenMAX IL i té tots els camps necessaris per tal de poder mantenir l’estat del component d’OpenMAX. Aquesta estructura és la següent:

```
typedef struct OmxContext{

        OMX_HANDLETYPE hComp; //component

        OMX_STATETYPE eState; //Estat

        OMX_U32 nPorts; //Numero de ports

        FILE *pOutputFile; //Punter al fitxer de entrada

        FILE *pInputfile; //Punter al fitxer de sortida

        BufferList *pInBufferList; //Punter al buffer de entrada

        BufferList *pOutBufferList; // Punter al buffer de sortida

        OMX_BOOL bClientAllocBuf; //Boolean de configuració

} SampleCompTestCtxt;

typedef struct _BufferList BufferList;
```

```
struct _BufferList{

        OMX_BUFFERHEADERTYPE *pBufferHeader;

        BufferList *pNextBuffer;

};
```

Codi 18: Estructura d'OpenMAX

Com s'ha dit a l'apartat d'OpenMAX IL els components tenen estats, i per poder entendre la implementació d'aquesta API és clau saber com actuen aquest estats als components. Els components poden estar als següents estats:

WAIT FOR RESOURCES: El component està a l' espera d'assignar-li recursos per començar a treballar.

IDLE: El component ha sigut carregat amb tots els seus recursos, però encara no ha començat a processar cap dada, en aquest estat es pot accedir a tots els recursos i al buffer.

EXECUTING: El component està en execució processant i transferint dades.

PAUSED: El component està en pausa, però pot continuar processant al mateix punt on es va parar.

INVALID: El component ha trobat un error crític del qual no es pot recuperar. L' única manera de sortir d'aquest estat es tornar a carregar el component. En aquest estat es pot dir que no té cap recurs assignat i el buffer és inaccessible.

LOADED: El component ha sigut carregat però no se li ha assignat cap recurs, el buffer no és accessible en aquesta fase.

UNLOADED: El component no ha estat inicialitzat. Aquest és l'estat inicial de tots els components.

Al següent esquema es mostra les transicions entre els diferents estats.

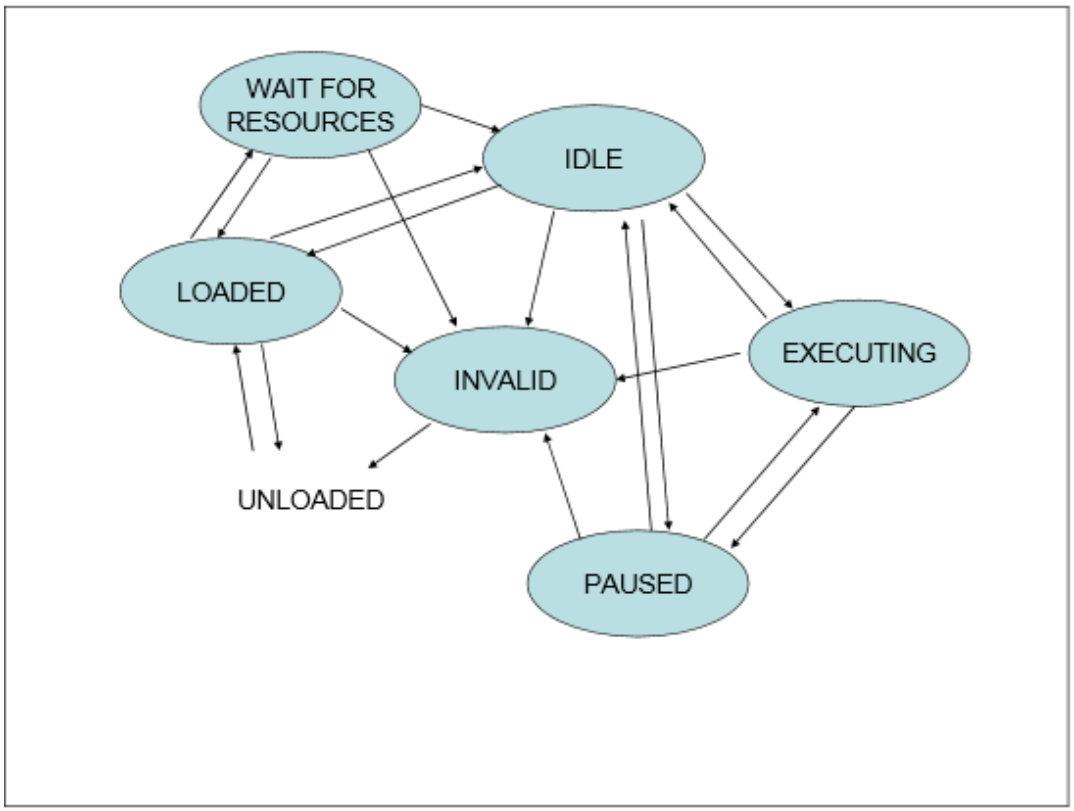

Figura 34: Diagrama d'estats d'OpenMAX IL

La implementació dels estats queden de la següent manera a la nostra aplicació.

```
OMX_STRING StateToString(OMX_STATETYPE eState) {

        OMX_STRING StateString;

        switch (eState) {

        case OMX_StateInvalid:

                StateString = "Invalid";

                break;

        case OMX_StateLoaded:

                StateString = "Carregat";
```

```
                break;
        case OMX_StateIdle:
                StateString = "En espera";
                break;
        case OMX_StateExecuting:
                StateString = "Executant";
                break;
        case OMX_StatePause:
                StateString = "Pausat";
                break;
        case OMX_StateWaitForResources:
                StateString = "Esperant per recursos ";
                break;
        default:
                StateString = "Desconegut";
                break;
        }
        return StateString;
}
```

Codi 19: Implementació dels estats

Recordarem com funciona la comunicació entre els components. La comunicació entre components ha de ser mitjançant ports. La seqüència de comunicació ha de ser la següent:

- Es crida la funció OMX_getHandle, aquesta funció crea el component i aquest component passa a estat LOADED.
- A continuació el nucli de OMX IL ha de passar al component les funcions mitjançant la funció Set Callbacks.
- Configurar el component i els ports de comunicació que seran utilitzats amb la funció OMX_SetParameters.
- Després de la configuració es pot dir al component que passi a l'estat IDLE amb la funció OMX_StateIdle.

Una vegada a l'estat IDLE es poden configurar els buffers als ports. Per dur a terme aquesta operació s'utilitza les funcions OMX_AllocateBuffer i OMX_UseBuffer.

Una vegada fet això ja es pot passar a estat EXECUTING mitjançant la funció OMX_stateExecuting, aquest component transmet les dades al buffer i amb la funció OMX_EmptyThisBuffer es pot accedir a les dades.

Al finalitzar la tasca el component ha d'entrar al estat IDLE per poder deixar tots els buffer lliures (OMX_FreeBuffer).

Per acabar nomes ens queda cridar a la funció OMX_FreeHandle per poder alliberar tots els components. Aquesta funció es important perquè pot provocar que no es pugui tornar a utilitzar una altra vegada aquest component.

Per poder canviar l'estat de manera rapida s'ha creat una funció amb visibilitat privada, només accessible des de la pròpia llibreria. Aquesta funció canvia l'estat del component.

```
OMX_ERRORTYPE change_state(OMX_STATETYPE eToState, OmxContext* pContext)

{

        OMX_ERRORTYPE eError = OMX_ErrorNone;

        OMX_PARAM_PORTDEFINITIONTYPE tPortDef;

        eError = OMX_SendCommand(pContext->hComp, OMX_CommandStateSet, eToState,
NULL);
```

```
__android_log_print(ANDROID_LOG_DEBUG, "TRY_OMX", “SendCommand %x", eError);

//en cas de voler canviar a estat LOADED a IDLE

        if ((eToState == OMX_StateIdle) && (pContext->eState == OMX_StateLoaded))

        {

                eError = OMX_GetParameter(pContext->hComp,
OMX_IndexParamPortDefinition,  (OMX_PTR) & tPortDef);

        __android_log_print(ANDROID_LOG_DEBUG,  "TRY_OMX",  “Get_parameter  %x”,
eError);

        eError =AllocateBuffers(pContext, &tPortDef);

        __android_log_print(ANDROID_LOG_DEBUG, "TRY_OMX", “Buffers allocats”);

        }

 //en cas de voler canviar a estat IDLE a LOADED

else if ((eToState == OMX_StateLoaded) && (pContext->eState == OMX_StateIdle))

        {

                eError = MemMgr_Free(pBuffer);

                pContext->pInBufferList = NULL;

                pContext->pOutBufferList = NULL;

                __android_log_print(ANDROID_LOG_DEBUG,        "TRY_OMX",        “Buffers
esborrats”);

        }

//comprovació de que tot ha funcionat be

        if (pContext->eState != eToState) {
```

```
        __android_log_print(ANDROID_LOG_DEBUG, "TRY_OMX", “Estat incorrecte %x", eError);

        }

        return eError;

}
```

Codi 20: Funció canvi d’estat

Una vegada que ja tenim les estructures creades i les funcions auxiliars implementades ja podem començar a implementar la nostre API de descodificació de vídeo. Tal i com s’ha dit en la fase de disseny la primera funció és “h264_create”, aquesta funció té com a tasca crear els components i la configuració per tal de poder dur a terme la descodificació.

```
OMX_ERRORTYPE h264_create(OmxContext* pContext){

        OMX_ERRORTYPE eError = OMX_ErrorNone;

        OMX_HANDLETYPE hComp = NULL;

        OMX_CALLBACKTYPE oCallbacks;

        OmxContext*pContext = NULL;

        OmxContextoAppData;

        memset(pContext, 0x0, sizeof(pContext));

        oCallbacks.EventHandler = SampleTest_EventHandler;

        oCallbacks.EmptyBufferDone = SampleTest_EmptyBufferDone;

        oCallbacks.FillBufferDone = SampleTest_FillBufferDone;

        //creació del semafors

        TIMM_OSAL_SemaphoreCreate(&pContext->hStateSetEvent, 0);
```

```
        TIMM_OSAL_SemaphoreCreate(&pContext->hPortDisableEvent, 0);

        eError = OMX_Init();

        __android_log_print(ANDROID_LOG_DEBUG, "TRY_OMX", “OMX_Init() %x", eError);

        eError = OMX_GetHandle(&hComp, (OMX_STRING) COMPONENT_NAME,
pContext, &oCallbacks);

        __android_log_print(ANDROID_LOG_DEBUG, "TRY_OMX", “OMX_GetHandle %x",
eError);
//Comprovació que el component estigui en l’estat adequat

        eError = OMX_GetState(pContext->hComp, &pContext->eState);

        if (OMX_StateLoaded != pContext->eState) {

        __android_log_print(ANDROID_LOG_DEBUG, "TRY_OMX", “Estat incorrecta %x",
eError);

        }
//detectar els ports de vídeo del component

        eError = OMX_GetParameter(hComp, OMX_IndexParamVideoInit, (OMX_PTR) &
pContext->sPortParam[1]);

        __android_log_print(ANDROID_LOG_DEBUG, "TRY_OMX", “GetParameter %x",
eError);
//detectar els port de imatge del component

        OMX_TEST_INIT_STRUCT(pContext->sPortParam[2], OMX_PORT_PARAM_TYPE);

        eError = OMX_GetParameter(hComp, OMX_IndexParamImageInit,

        (OMX_PTR) & pContext->sPortParam[2]);
```

```
        __android_log_print(ANDROID_LOG_DEBUG, "TRY_OMX", “GetParameter %x", eError);

//actualitzar el camp nPorts

        pContext->nPorts = pContext->sPortParam[0].nPorts + pContext->sPortParam[1].nPorts + pContext->sPortParam[2].nPorts + pContext->sPortParam[3].nPorts;

        //El component allocata el buffer

        pContext->bClientAllocBuf == OMX_FALSE;

return eError;

}
```

Codi 21: Funció create (crear el component)

A continuació la implementació de l’operació de descodificar, aquesta funció també és la que configura tots el paràmetres corresponents amb l’estàndard H264.

```
OMX_ERRORTYPE h264_decode(OmxContext* pContext, string rutaFile) {

//obtindrà paràmetres dels ports

        tPortDef.nPortIndex = OMX_VIDEO_DECODE_INPUT_PORT;

        eError = OMX_GetParameter(pContext->hComp, OMX_IndexParamPortDefinition, (OMX_PTR) & tPortDef);

        __android_log_print(ANDROID_LOG_DEBUG, "TRY_OMX", “GetParametre %x", eError);

        tPortDef.format.video.eDir = OMX_DirInput;

        tPortDef.format.video.bEnabled = OMX_TRUE;

        tPortDef.format.video.nFrameWidth = amplada(&rutaFile);
```

```
        tPortDef.format.video.nFrameHeight = altura (&rutaFile);
        tPortDef.format.video.nBufferSize =  amplada(&rutaFile) * altura (&rutaFile) * 3)/2;
        tPortDef.format.video.eCompressionFormat = OMX_VIDEO_CodingAVC;
        tPortDef.format.video.eColorFormat = OMX_COLOR_FormatUnused;
        eError = OMX_SetParameter(pContext->hComp, OMX_IndexParamPortDefinition,
(OMX_PTR) & tPortDef);
        __android_log_print(ANDROID_LOG_DEBUG, "TRY_OMX", “SetParameter %x", eError);
//obté el OUTPUT paràmetres per defecte i modifiquem els camps
        tPortDef.nPortIndex = OMX_VIDEO_DECODE_OUTPUT_PORT;
        eError = OMX_GetParameter(pContext->hComp, OMX_IndexParamPortDefinition,
(OMX_PTR) & tPortDef);
        __android_log_print(ANDROID_LOG_DEBUG, "TRY_OMX", “GetParametre %x",
eError);
        tPortDef.format.video.eDir = OMX_DirInput;
        tPortDef.format.video.bEnabled = OMX_TRUE;
        tPortDef.format.video.nFrameWidth = amplada(&rutaFile);
        tPortDef.format.video.nFrameHeight = altura (&rutaFile);
        tPortDef.format.video.nBufferSize =  amplada(&rutaFile) * altura (&rutaFile) * 3)/2;
        tPortDef.format.video.eCompressionFormat = OMX_COLOR_FormatUnused;
        tPortDef.format.video.eColorFormat = OMX_COLOR_FormatYUV420Planar;
        eError = OMX_SetParameter(pContext->hComp, OMX_IndexParamPortDefinition,
(OMX_PTR) & tPortDef);
        __android_log_print(ANDROID_LOG_DEBUG, "TRY_OMX", “SetParameter %x", eError);
```

```
        pContext->pInputfile = fopen(rutaFile, "rb");
        if (NULL == pContext->pInputfile) {
                eError = OMX_ErrorInsufficientResources;
                __android_log_print(ANDROID_LOG_DEBUG, "TRY_OMX", “Change state %x",
eError);
        }
        pContext->pOutputFile = fopen(NON_TUN_OUTPUT_FILE, "wb");
        if (NULL == pContext->pOutputFile) {
                eError = OMX_ErrorInsufficientResources;
                __android_log_print(ANDROID_LOG_DEBUG, "TRY_OMX", “AL obrir fitxer %x",
eError);
        }
        __android_log_print(ANDROID_LOG_DEBUG, "TRY_OMX", “Fitxer obert”);

        //Idle
        eError = change_state(OMX_StateIdle, pContext);
        __android_log_print(ANDROID_LOG_DEBUG, "TRY_OMX", “Change state %x", eError);
        //Executing
        eError = change_state (OMX_StateExecuting, pContext);
        __android_log_print(ANDROID_LOG_DEBUG, "TRY_OMX", “Change state %x", eError);
        //Processament dels buffers
        eError = WriteInBuffers(pContext);
```

```
        __android_log_print(ANDROID_LOG_DEBUG, "TRY_OMX", “Write buffer %x", eError);

                eError = ReadOutBuffers(pContext);

        __android_log_print(ANDROID_LOG_DEBUG, "TRY_OMX", “Write buffer %x", eError);

return eError;

}
```

Codi 22: Funció decode (descodifica)

A continuació es mostra el fragment de codi corresponent a la funció cancel·la que s'encarrega de cancel·lar l'operació de descodificar i tornar a deixar el buffers preparats per a una altra tasca de descodificació.

```
OMX_ERRORTYPE h264_cancel(OmxContext* pContext){

        eError = sendCommand(pContext,OMX_CommandFlush, pContext->pInBufferList);

        eError = sendCommand(pContext,OMX_CommandFlush, pContext-> pOutBufferList);

        __android_log_print(ANDROID_LOG_DEBUG, "TRY_OMX", “Error al buidar el buffers %x", eError);

//Tornar a l'estat espera

        eError = change_state (OMX_StateIdle, pContext);

        __android_log_print(ANDROID_LOG_DEBUG, "TRY_OMX", “Estat en espera %x", eError);

        return eError;

}
```

Codi 23: Funció cancel (cancel·la)

Per últim, només ens cal veure l' operació de destrucció dels components i de les interfícies i tancar tots el possibles fitxers oberts durant el procés. Aquesta operació és molt important per deixar el dispositiu consistent per a posteriors tasques.

```
OMX_ERRORTYPE h264_destroy(OmxContext* pContext) {

//Tornar a l'estat carregat

        eError = change_state (OMX_StateLoaded, pContext);

        __android_log_print(ANDROID_LOG_DEBUG, "TRY_OMX", "Estat en espera %x", eError);

//tanca els fitxers de entrada i/o sortida si ni han

        if (pContext->pInputfile){

                fclose(pContext->pInputfile);

        }

        if (pContext->pOutputFile){

                fclose(pContext->pOutputFile);

        }

        if (eError == OMX_ErrorNone){

                eError = OMX_FreeHandle(pContext->hComp);

                __android_log_print(ANDROID_LOG_DEBUG, "TRY_OMX", "Free_handle %x", eError);

                eError = OMX_Deinit();

                __android_log_print(ANDROID_LOG_DEBUG, "TRY_OMX", "Deinit %x", eError);

        }
```

```
        else{
        __android_log_print(ANDROID_LOG_DEBUG, "TRY_OMX", “No s’ha pogut destruir el component”);
        }
        //destrucció dels semàfors creats
        TIMM_OSAL_SemaphoreDelete(pContext->hStateSetEvent);
        TIMM_OSAL_SemaphoreDelete(pContext->hPortDisableEvent);
        __android_log_print(ANDROID_LOG_DEBUG, "TRY_OMX", “Destrucció dels semàfors correcta”);
        return eError;
}
```

Codi 24: Funció destroy (destrueix)

A continuació es mostran altres funcions auxiliares de visibilitat privada utilitzades per completar el desenvolupament d’aquesta API. Aquestes funcions auxiliars són les que es basen en la escriptura i lectura dels buffers.

```
OMX_ERRORTYPE WriteInBuffers (OmxContext* pContext) {
        OMX_ERRORTYPE eError = OMX_ErrorNone;
        BufferList *pList = NULL;
        OMX_BUFFERHEADERTYPE *pBufHeader = NULL;
        pList = pContext->pInBufferList;
        while (pList && pList->pBufferHeader) {
                BUFFERLIST_CLEAR_ENTRY(pList, pBufHeader);
```

```
                pTmpBuffer =TIMM_OSAL_Malloc(OMX_SAMPLE_BUFFER_SIZE, 0, 0, 0);
                if (pTmpBuffer == NULL)
                __android_log_print(ANDROID_LOG_DEBUG, "TRY_OMX", “Insuficient recursos”);
                pOrigTmpBuffer = pTmpBuffer;
                ReadInputFile(pContext, pTmpBuffer, OMX_SAMPLE_BUFFER_SIZE, pContext->pInputfile);
                pBufHeader->nFilledLen = OMX_SAMPLE_BUFFER_SIZE;
                TIMM_OSAL_Free(pOrigTmpBuffer);
                ReadInputFile(pContext, pBufHeader->pBuffer,
                 pBufHeader->nAllocLen, pContext->pInputfile);
                pBufHeader->nFilledLen = pBufHeader->nAllocLen;
                if (pContext->bEOS == OMX_TRUE)b{
                        pBufHeader->nFlags |= OMX_BUFFERFLAG_EOS;
                }
                eError = OMX_EmptyThisBuffer(pContext->hComp, pBufHeader);
        __android_log_print(ANDROID_LOG_DEBUG, "TRY_OMX", “Buidar dels buffers %x”,eError);
        }
return eError;
}
```

Codi 25: Funció WriteInBuffers *(Escriptura en buffer)*

```
OMX_ERRORTYPE ReadOutBuffers(OmxContext* pContext) {

        OMX_ERRORTYPE eError = OMX_ErrorNone;

        BufferList *pList = NULL;

        OMX_BUFFERHEADERTYPE *pBufHeader = NULL;

        pList = pContext->pOutBufferList;

        while (pList && pList->pBufferHeader) {

        BUFFERLIST_CLEAR_ENTRY(pList, pBufHeader);

        eError = OMX_FillThisBuffer(pContext->hComp, pBufHeader);

        __android_log_print(ANDROID_LOG_DEBUG, "TRY_OMX", “Emplena els buffers %x”,eError);

        if (pBufHeader->nFlags == OMX_BUFFERFLAG_EOS) {

                pContext->nBufDoneCalls = OMX_TEST_BUFFERS_OF_TRAFFIC;

        }

        __android_log_print(ANDROID_LOG_DEBUG, "TRY_OMX", “Emplena els buffers %x”,eError);

        return eError;

}
```

Codi 26: Funció ReadOutBuffers *(Lectura del buffer)*

Amb això queda tota la nostra API de descodificació implementada. Una vegada compilada ens queda una biblioteca .so dinàmica amb les funcions descrites anteriorment. Al següent apartat ens centrarem en testejar aquesta API.

5.3 Proves

A aquest apartat es faran 2 tipus de proves. Les primeres per comprovar que es compleixen les funcionalitats descrites i les segones per comprovar les nostres millores respecte a la descodificació per software i veure si hem aconseguit el mateix rendiment que desenvolupant l'aplicació a una capa superior d'OpenMAX.

Per fer les proves de complir les funcionalitats descrites utilitzarem la interfície creada als apartats anteriors. A part també comprovarem els logs creats al transcurs de l' execució de l'aplicació, per assegurar-nos que no s'ha donat cap error.

Després de provar amb els vídeos de mostra podem assumir que l'aplicació amb la nostra biblioteca funciona correctament.

Veient el log d'Android es pot observar que dóna el error 0x8000100 a l'hora de descodificar els primers píxels. Aquest error es degut a que el component no té suficients recursos, tot i a aquest error, la descodificació del vídeo a simple vista sembla correcta. Com a simple vista no podem assegurar al 100% que la descodificació sigui correcta i a més a més degut a aquest error, s'intueix que pot haver-hi alguna cosa que no funciona, s'ha descompost el fitxer en bytes per tal d'analitzar frame a frame a nivell de bytes si fa la descodificació correcta. Després de fer aquesta comparació s'ha trobat que els primers 3 frames són incorrectes, és a dir, el protocol comença bé amb la seqüència 00 00 01. Però després hi ha errors als següents frames. A partir del 4 frame el protocol funciona correctament. És per aquest motiu que al ser només 3 o 4 frames incorrectes no es veu l'error a simple vista.

A continuació recordem els tres vídeos de diferents qualitats.

Nom	Ample	Alçada	Format	FPS	% CPU
Vídeo 1	160	122	H264	17,84	9,10%
Vídeo 2	800	600	H264	18,66	13,76%
Vídeo 3	1920	1088	H264	8,06	15,54%

Taula 16: Dades extretes de la biblioteca desenvolupada

	Mostra 1		Mostra 2		Mostra 3		Mostra 4		Mostra 5	
Nom	**FPS**	**% CPU**	**FPS**	**% CPU**	**FPS**	**% CPU**	**FPS**	**% CPU**	**FPS**	**% CPU**
Vídeo 1	17,40	9,10%	18,20	8,90%	18,30	9,40%	17,50	9,10%	17,80	9,00%
Vídeo 2	17,50	13,80%	17,40	14,40%	18,40	13,20%	18,30	13,30%	21,70	14,10%
Vídeo 3	8,30	14,40%	8,50	14,90%	7,40	15,10%	9,10	18,00%	7,00	15,30%

Taula 17: Dades extretes de la biblioteca desenvolupada

Podem apreciar que no hem arribat als valors de fps que havíem aconseguit anteriorment, però aquests valors no són significatius.

Pel que fa a l'ús de la CPU s'ha mantingut als mateixos valors que la aplicació desenvolupada amb OpenMAX AL. Als següent gràfics es mostren la comparació entre les 3 aplicacions.

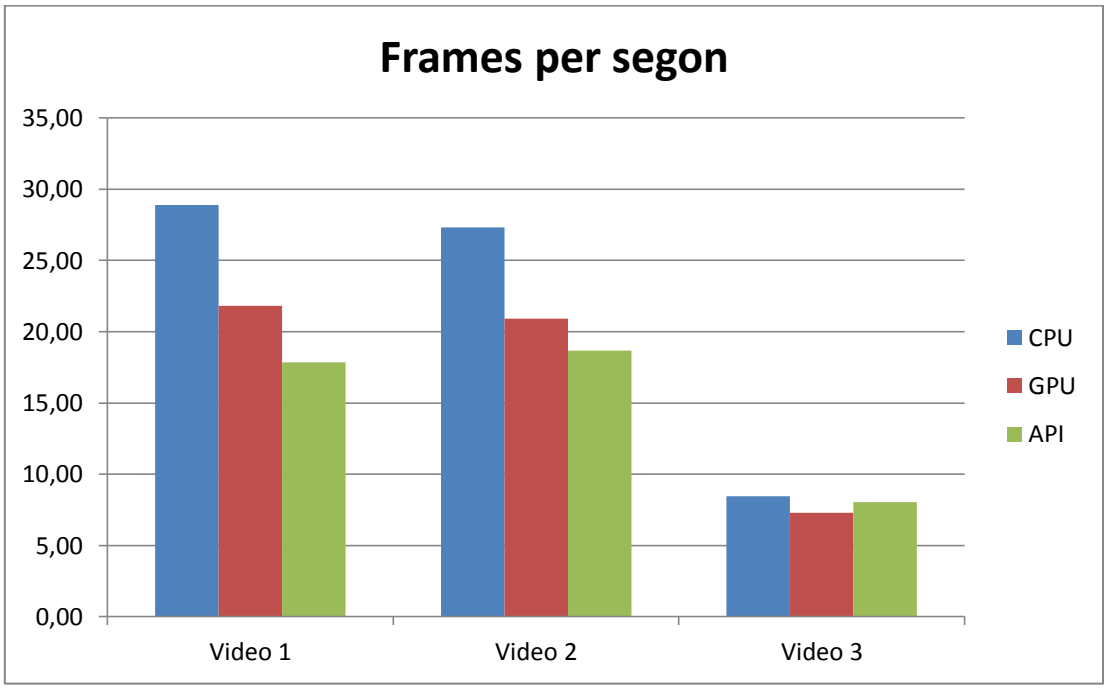

Figura 35: Gràfic comparatiu de consum de la CPU entre les 3aplicacions desenvolupades

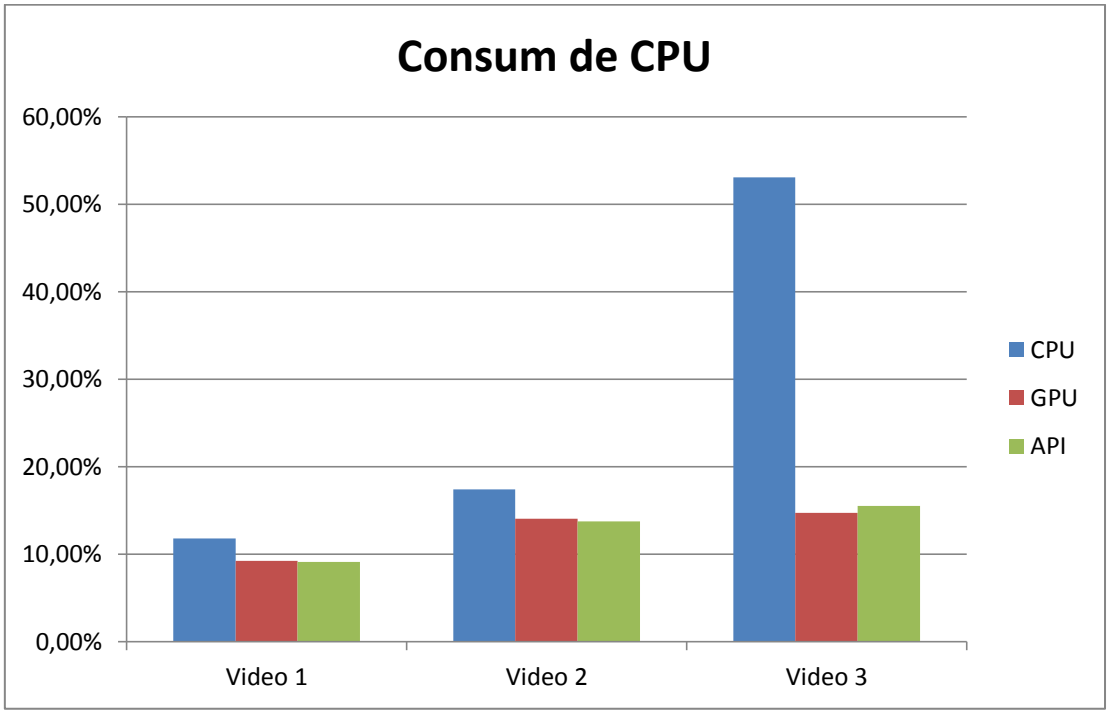

Figura 36: Gràfic comparatiu de frames per segon entre les 3 aplicacions desenvolupades

6. Conclusions

Una vegada acabat tot el projecte és el moment de treure conclusions sobre el tema i opinions personals.

Pel que fa al projecte, crec que s'ha aconseguit complir tots els objectius marcats, a més totes les proves i experiments realitzes han aconseguit confirmar les teories i suposicions inicials. S'ha aconseguit fer una biblioteca per a que desenvolupadors puguin utilitzar la unitat de processament gràfica. Tot i que en aquest moment ja entenc com funciona OpenMAX i està tota la biblioteca desenvolupada, és una llàstima no tenir més temps per invertir en desenvolupar més funcionalitats, donat que ara és relativament senzill i ràpid anar afegint funcionalitats.

A nivell personal em sento orgullós de la feina realitzada i penso que he aconseguit els objectius que m'havia proposat inicialment. M'he adonat també, que les aportacions obtingudes no han sigut solament a nivell acadèmic sinó a nivell personal i m'ha servit per encoratjar-me a continuar estudiant al camp de dispositius mòbils i multimèdia.

Haig de dir, però, que malgrat les dificultats sorgides al seu procés i el temps invertit, em sento satisfet amb la culminació d'aquest projecte.

7. Treball futur

En aquest apartat es farà una breu explicació de com es pot arribar a ampliar aquest projecte. Tal i com s'ha comentat al disseny de la nostra API de descodificació de vídeo, ja s'han tingut en compte les futures possibles ampliacions d'aquesta.

La primera ampliació possible és adaptar-la a més dispositius, tot i que inicialment es volia treballar amb tots als dispositius, es va haver d'elegir un sol dispositiu per fer la implementació. Anàlogament com mostra aquest document es pot utilitzar per fer els mateixos camins per uns altres fabricants i models (fins i tot per models futurs). Per tant aquesta API amb el temps es pot convertir en tot un model de referència per tots els dispositius del mercat.

Paral·lelament a fer la API compatible amb més dispositius i fabricants del mercat és pot incrementar els formats admesos per aquesta API. Com s'ha explicat en aquest document ens hem centrat exclusivament en el format H264, per tant es podria treballar en aconseguir la descodificació de més formats existents al mercat.

Una altra rama d'ampliació d'aquest treball es afegir-li tota la part de so al vídeo. Com s'ha dit al principi d'aquest document el so el deixàvem de banda per motius de temps i ens centraríem únicament en el vídeo.

Una altra ampliació possible és fer la nostra API compatible amb més sistemes operatius, és a dir fer que es pugui utilitzar en sistemes que treballin en entorn Windows o Mac.

Tot aquestes ampliacions són pràcticament infinites ja que de formats de vídeo nous, de dispositius de fabricants i de sistemes operatius apareixen cada any centenars d'ells.

A més cal dir que es pot dedicar més hores a la optimització de codi per aconseguir millor rendiment.

Per últim, remarcarem la possibilitat d' adaptar la nostra API per tal de que es pugui codificar vídeo. Aquesta és una ampliació que s'ha estudiat durant el transcurs d'aquest projecte tot i que mai s'ha arribat a implementar. Per codificar vídeo es necessita seguir els mateixos passos, és a dir, crear el component, configurar els ports, canviar a l'estat executant i destruir els components creats. Així que podem assumir que els passos a seguir són els mateixos. A

continuació es mostrarà un breu resum dels passos necessaris per adaptar aquesta API per poder codificar vídeo.

1. Crear el component h264, dlopen, init, gethandle...
2. Configurar els paràmetres (setParametre) del ports de entrada, a continuació es mostren aquest paràmetres:

   nFrameWidth = amplada

   nFrameHeight = llargada

   sOmxInPortDef.nBufferSize= (amplada *llargada * 3)/2;

   sOmxInPortDef.eDir = OMX_DirInput;

   sOmxInPortDef.bEnabled = OMX_TRUE;

   nStride = 320;

   eColorFormat = OMX_TI_COLOR_FormatYUV420PackedSemiPlanar;

3. Configurar els paràmetres (setParametre) dels ports de sortida.

   .eDir = OMX_DirInput;

   bEnabled = OMX_TRUE;

   video.eCompressionFormat = OMX_VIDEO_CodingAVC;

   .nBufferSize = (amplada *llargada* 3)/2;

4. Configurar el bitrate, per exemple, 1Mbps
5. Obtenir els paràmetres (cridan OMX_IndexParamVideoAvc) i modificant els següents camps:

   sAvcParam.eProfile = OMX_VIDEO_AVCProfileBaseline;

   sAvcParam.nPFrames = 30;

sAvcParam.nBFrames= 0;

sAvcParam.bDirect8x8Inference= 0;

sAvcParam.bEntropyCodingCABAC = 0;

sAvcParam.nCabacInitIdc = 0;

sAvcParam.bEnableFMO= 0;

sAvcParam.bEnableASO = 0;

sAvcParam.bEnableRS = 0;

sAvcParam.nSliceHeaderSpacing= 0;

sAvcParam.bFrameMBsOnly= 1;

sAvcParam.bMBAFF = 0;

sAvcParam.bWeightedPPrediction= 0;

6. Canviar a l'estat IDLE
7. Fer el allocate dels buffers d' entrada i de sortida
8. Canviar a l'estat executant i començar la codificació.
9. Destruir els component creats

8. Bibliografia

http://www.khronos.org/openmax/

http://en.wikipedia.org/wiki/OpenMAX

http://mirror.yongbok.net/pub/linux/android/repository/ndk/docs/openmaxal/

http://developer.android.com/index.html

http://developer.android.com/tools/sdk/ndk/index.html#Using

http://processors.wiki.ti.com/index.php/OpenMax_Development_Guide

http://trivedihardik.wordpress.com/2011/06/16/hello-world-example-using-ndk-in-android/

http://e2e.ti.com/support/default.aspx

http://omappedia.org/wiki/Ducati_For_Dummies

http://stackoverflow.com/

http://ffmpeg.org/trac/ffmpeg/wiki/x264EncodingGuide

http://vec.io/posts

http://www.tr.ietejournals.org/article.asp?issn=0256-4602;year=2011;volume=28;issue=2;spage=146;epage=157;aulast=Ram%EDrez-Acosta

http://www.ffmpeg.org/

http://www.roman10.net/how-to-build-ffmpeg-for-android/

http://dranger.com/ffmpeg/

https://groups.google.com/forum/#!overview

http://www.ti.com/tool/androidsdk-dm37x

http://ccppcoding.blogspot.com.es/2012/09/openmax.html

http://omxil.sourceforge.net/

http://elinux.org/images/5/52/Elc2011_garcia.pdf

http://gentlelogic.blogspot.com.es/2011/11/exploring-h264-part-2-h264-bitstream.html

http://alucard1990.hubpages.com/hub/How-to-Make-a-Simple-Media-Player-for-Android

http://src.chromium.org/svn/trunk/src/third_party/openmax/il/OMX_Core.h

http://pubs.opengroup.org/onlinepubs/7908799/xsh/dlfcn.h.html

http://developer.apple.com/library/mac/#technotes/tn2267/_index.html

http://www-ee.uta.edu/dip/courses/ee5356/H264systems.pdf

https://developer.nvidia.com/

http://www.ti.com/lit/wp/spry193/spry193.pdf

http://stackoverflow.com/questions/12817198/ffmpeg-1-0-android-ndk-r8b

http://ikaruga2.wordpress.com/2011/06/15/video-live-wallpaper-part-2/

http://libav-users.943685.n4.nabble.com/Libav-user-Speedup-FFmpeg-h264-Decoding-on-Android-Devices-td4655525.html

http://stackoverflow.com/questions/5164879/ffmpeg-play-video-problem-too-slow-or-too-fast-playing-video-at-right-peed

http://www.arm.com/community/multimedia/standards-apis.php

http://www.cnx-software.com/tag/openmax/

http://eon.sdsu.edu/~kumar/sites/default/files/IETETechRev_M2_2011.pdf

9. Glossari

A aquest apartat s'intentarà definir els tecnicismes que s'han utilitzat al llarg del document, aquestes normalment són paraules adoptades de l'anglès.

Frame

El quadre (s'utilitza el terme en Anglès frame) és la unitat mínima, en forma d'imatge estàtica, en què es pot descompondre una seqüència de vídeo.

Framework

L'entorn de treball (s'utilitza el terme en Anglès framework). Inclou tots aquells recursos i funcionalitats que determinen el desenvolupament organitzat d'un projecte.

Unitat central de processament (CPU)

És el component de l'ordinador i d'altres dispositius que interpreta les instruccions contingudes en aplicacions i processa les dades. És un component imprescindible a tots els dispositius mòbils i ordinadors.

Unitat de Procés Gràfic (GPU)

És un dispositiu dedicat a la generació de gràfics. Una GPU és capaç d'aplicar operacions sobre gràfics molt més ràpid que una CPU.

Software

El programari (s'utilitza el terme en Anglès software) és el conjunt dels programes informàtics, procediments i documentació que fan alguna tasca a un dispositiu.

Hardware

El maquinari (en anglès hardware) d'un ordinador o dispositiu és el conjunt de les seves parts físiques.

Còdec

Un còdec és un esquema que regula una sèrie de transformacions en un senyal o informació. Els còdecs tant poden transformar un senyal a una forma codificada com fer-ho al revés.

Interfície de Programació Aplicacions API

Una Interfície de Programació d'Aplicacions és un conjunt de declaracions que defineix el contracte d'un component informàtic amb qui farà ús dels seus serveis.

Frames per segon (FPS)

FPS vol dir el número d'imatges que es mostra per segon. La freqüència en que els fotogrames apareixen és proporcional al número de píxels que el dispositiu ha de generar i afecta directament al rendiment del dispositiu.

Per posar alguns exemples, les GPU actuals d'un ordinador de gamma mitjana-alta poden arribar a 100fps. Ara bé, la qüestió important és quan deixa l'ull humà de notar la fluïdesa de la imatge? La resposta depèn de varis factors: Si per exemple estem parlant d'un vídeo amb poc moviment (una paret per exemple) amb 10 fps notarem el mateix efecte que amb 1000 fps. Les pel·lícules actualment s'emiteixen a 24 fps (l'ultimes del mercat a 48 fps) i l'ull humà es capaç de detectar el canvi. Resumint que potser una persona pot veure 120 fps o potser 200 fps, depèn de la persona en qüestió i del tipus d'imatge.

Càrrega del dispositiu

Calcularem el rendiment del dispositiu mitjançant la comanda top des de l'ordinador. El rendiment del dispositiu vindrà indicar per el % d'ús de la CPU, cal dir que aquesta dispositiu utilitzat per aquest projecte té 2 cores, per tant un 50% vol dir que estem treballant al màxim sense fer multithreating.

Hem de tenir molta cura que no hi hagi cap aplicació en segon pla activa que ens distorsioni la mesura del rendiment.

Qualitat de vídeo (resolució)

La resolució de la pantalla és el número de píxels que es pot mostrar a la pantalla. Ve donada per 2 números que indiquen l'amplada per l'alçada, aquest es mesurat en píxels. Una de les resolucions més baixes seria de 320×200, la més alta que s'utilitza en aquest projecte és de 1184x720. Cal dir que a més resolució més píxels s'han de dibuixar per frame, per tant més cost pel dispositiu.

10. Agraïments

Vull agrair la col·laboració de totes aquelles persones i institucions que han fet possible la realització d'aquest projecte.

Primer de tot a agrair a la meva família per donar-me tot el recolzament necessari, no només per realitzar aquest projecte si no per acabar tota la carrera.

Al meu tutor en Jordi que ha estat un any sencer guiant-m'hi per la senda correcta per la realització d'aquest projecte. En tot moment m'ha recolzat sempre sense ficar cap tipus de pressió.

Per l' ajuda de 2 professors de la UPC, en Xavi i l' Àlex, que amb els seus consells m'han donat la força necessària de saber que el que feia era possible.

Agrair també a l' empresa en la qual estic treballant, ja que sense la seva flexibilitat horària no hagués estat possible dedicar-li totes les hores que calen per finalitzar aquest projecte.

Agrair a tots els amics per la paciència que heu tingut quan m'esborrava dels plans per poder continuar amb el projecte. I en especial a la Maria Jose que és la que més ha patit en aquest projecte.

I per últim agrair a aquelles persones anònimes que amb els seus comentaris i suport a Internet heu aconseguit que pugui resoldre tots els problemes que m'he anat trobant.